ONLINE LEARNING ALGORITHMS FOR SEQUENCE PREDICTION, IMPORTANCE WEIGHTED CLASSIFICATION, AND ACTIVE LEARNING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Nikolaos Karampatziakis

August 2012

ONLINE LEARNING ALGORITHMS FOR SEQUENCE PREDICTION, IMPORTANCE WEIGHTED CLASSIFICATION, AND ACTIVE LEARNING

Nikolaos Karampatziakis, Ph.D.

Cornell University 2012

This thesis studies three problems in online learning. For all the problems the proposed solutions are simple yet non-trivial adaptations of existing online machine learning algorithms. For the task of sequential prediction, a modified multiplicative update algorithm that produces small and accurate models is proposed. This algorithm makes no assumption about the complexity of the source that produces the given sequence. For the task of online learning when examples have varying importances, the proposed algorithm is a version of gradient descent in continuous time. Finally, for the task of efficient online active learning, the implementation we provide makes use of many shortcuts. These include replacing a batch learning algorithm with an online one, as well as a creative use of the aforementioned continuous time gradient descent to compute the desirability of asking for the label of a given example. As this thesis shows, online machine learning algorithms can be easily adapted to many new problems.

BIOGRAPHICAL SKETCH

Nikos Karampatziakis received a Ptychion from the Department of Informatics and Telecommunications at the University of Athens, Greece in 2003. In 2005 he received a Masters Degree from the same institution. Before enrolling at Cornell University in 2006, he spent a year working on a computer vision project jointly developed by the University of Athens and the University of Houston.

To my family.

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor Dexter Kozen. He helped me in many ways during my time here at Cornell and I am very grateful for that. I would also like to thank my committee members Thorsten Joachims, Giles Hooker, and Robert Kleinberg, for many very helpful discussions. Finally, I am extremely grateful to John Langford who gave me a chance to intern with him at Yahoo! Research in the summer of 2010 and has been helping me ever since.

Outside work, I have made many friends during my studies at Cornell. First there are the friends who entered the PhD program at around the same time as me: Lakshmi Ganesh, Sucheta Soundarajan, Hyung Chan An, Jesse Simons, Ymir Vigfusson, and Shaomei Wu. Next are the machine learning students: Tom Finley, Filip Radlinski, Alex Niculescu, Dan Sheldon, Yisong Yue, Art and Katy Munson, Daria and Alex Sorokin, and Pannaga Shivaswamy. Finally, I have made many Greek friends including Georgios Piliouras, Vasilis Syrgkanis, Theo Damoulas, Elias Bilionis, Demosthenis and Angela Chronis, Panos Athanasopoulos, Rebekka Christopoulou, Yannis Kamarianakis, Greg and Matoula Halkiopoulos, Eliza Kozyri, Vasilis Fountoulakis, Dimitra Bouziou, Kostas Mamouras, Panos Dallas, Stavros Nikolaou, Christina Argyrou, Georgios Oikonomou, and Apostolis Enotiadis.

Last but not least, I would like to thank Ainur Yessenalina for her love, care, and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis presents solutions to three core tasks in machine learning and focuses in particular on online learning. The first task has to do with sequential prediction and with how a learning algorithm can decide on a good representation of the examples it processes. In this context, an (infinite) hierarchy of features can be defined, and the features of each example form a path starting at the root of this hierarchy. The task of the learner is to estimate a model that both predicts well and uses few features. The second task is to carefully handle examples with weights that quantify relative importance. An *importance* is a real number that specifies the relative importance of each example. Examples with importances appear in many settings in machine learning but a popular online learning algorithm, *online gradient descent*, is very sensitive to large importances. The third task is that of active learning. In active learning, the learner can decide whether to ask for the label of the current example. The goal of the learner is to simultaneously find a model that generalizes well on unseen data and to minimize the number of requested labels. In the next sections, an overview of online learning and the three problems addressed in this thesis are presented.

1.1 Online Learning

Online learning is a machine learning protocol in which the learning algorithm receives examples one at a time. The learner then makes a prediction and receives some feedback before the next example is revealed. Online learning is becoming increasingly popular in machine learning research for many reasons: the relative ease of analyzing and implementing an online algorithm, the ability to provide worst case guarantees, and the interactive nature of the online pro-

tocol. Online learning finds many applications in the world wide web where users can interact with the information presented to them in the form of web search results, ads, or movie recommendations.

Online learning is primarily about modeling sequential decision-making, collecting and integrating new information, and providing worst case guarantees. However it can also be viewed as a strategy for implementing learning algorithms that work in the stochastic statistical setting. This view has its own merit. Many statistical machine learning tasks boil down to minimization of the empirical loss of a model under a constraint on the *complexity* of that model. Such tasks can often be posed as solutions of convex optimization problems. These optimization problems can be solved by the same algorithms that work for online learning and these algorithms are essentially optimal [56, 1]. For this reason, implementing statistical learning as a reduction to online learning has become the method of choice when working with datasets that are too big to fit in memory.

1.2 Contributions of this Thesis

The contributions of this thesis are online learning algorithms that can deal with

- a potentially infinite hierarchy of features,
- examples with importance weights,
- settings where labeling cost/effort is to be minimized.

Though the results are presented in the context of specific applications such as sequential prediction and importance-weighted classification, the developed algorithms are general and can work in other similar applications.

1.2.1 Online Learning of Prediction Suffix Trees

For the problem of sequential prediction, the main contribution of this thesis is an online learning algorithm that is *non-parametric*, meaning that the number of parameters that the algorithm will use will depend on the difficulty of the specific problem instance it will be executed on. The presented algorithm learns Prediction Suffix Trees without any a priori bound on their size. Prediction Suffix Trees (PSTs) are described in more detail in Sections 2.5 and 3.2.2. They are a popular tool for modeling sequences and have been successfully applied in many domains such as compression and language modeling. Here the well studied Winnow algorithm is adapted to the task of learning PSTs. The proposed algorithm automatically grows the tree so that it provably remains competitive with any fixed PST determined in hindsight. At the same time it is shown that the depth of the tree grows only logarithmically with the number of mistakes made by the algorithm. If there is a small PST that predicts the sequence without any mistakes, the algorithm will produce a PST of bounded size regardless of the length of the sequence. Finally, the effectiveness of this algorithm is empirically demonstrated in two different tasks.

1.2.2 Online Learning with Importance Weighted Examples

When an example is associated with an importance of i, this by definition means that it is as if the example appears i times in the data. More generally, an importance quantifies the relative importance of examples. Importances come up in many applications such as boosting [30], learning reductions [6], and active learning [4, 5]. However, a popular online learning algorithm, *online gradient descent*, is very sensitive to the way large importances are handled.

A naive approach would incorporate the importance by multiplication with

the gradient. However this approach has some difficulties. We argue that when importance weights are large, more sophisticated ways for dealing with them are necessary. We then develop an approach which enjoys an invariance property: that updating with importance weight $a + b$ is equivalent to updating once with importance weight a followed by another update with importance weight b. Even though this approach involves the solution of an ordinary differential equation, for many important loss functions this has a closed form update which satisfies standard regret guarantees when all examples have importance weight 1. Another reasonable approach for handling large importances, related to [47], is briefly discussed. Empirically, these approaches yield substantially superior prediction with similar computational performance while reducing the sensitivity of online gradient descent to the exact setting of the learning rate.

1.2.3 Online Active Learning

For many real world machine learning applications, labeled examples are scarce while unlabeled examples abound. It is also known [22] that active learning, allowing the learning algorithm to ask for the labels of specific examples, can reduce the amount of labels exponentially in certain cases. Many heuristics have been designed to decide when to ask for a label [50, 18, 60, 19]. However, when the problem is not realizable, i.e. when the best model in the class of models the learning algorithm is considering has an error rate greater than 0, these heuristics can completely fail to find the best model. This can happen because these heuristics can lead to biased sampling and the algorithm may not be collecting information that is representative of the underlying distribution [22].

In this thesis, an efficient online active learning algorithm is developed that is theoretically sound in an agnostic setting, empirically effective, and as efficient

as standard online learning algorithms. This algorithm is enabled by the updates introduced in Chapter 4, which carefully handle the large importances assigned to the examples by the active learning algorithm. It is empirically shown that when these importances are not handled carefully, the resulting algorithm is no better than collecting labels at random, also known as passive learning.

1.3 Thesis Outline

This thesis is organized as follows. Some background on online learning, active learning, and prediction suffix trees is given in Chapter 2. Chapter 3 describes an algorithm for learning small and accurate prediction suffix trees. Chapter 4 presents modifications of online gradient descent to handle large importance weights. In Chapter 5 an online active learning algorithm is developed that provably works without any assumptions on the data distribution. The thesis concludes and presents some directions for future work in Chapter 6.

CHAPTER 2

BACKGROUND

This thesis focuses on online learning algorithms. A learning algorithm is any algorithm that observes some data as its input and outputs a function that tries to explain the observed data. Online learning algorithms have to respond online, outputting such functions throughout the learning process. Of course there are many choices regarding what kind of functions the learning algorithm can output, how to measure how well a learning algorithm performs, and the learning algorithms themselves. The next sections describe these aspects. Finally, this chapter reviews some relevant work related to the specific problems discussed in the rest of this thesis.

2.1 Notation

Symbols typeset in normal font such as y will denote scalars. Vectors will be typeset in bold such as $\boldsymbol{w}$. Unless noted otherwise, vectors will belong to the d-dimensional vector space $\mathbb{R}^d$. The i-th element of vectors $\boldsymbol{u}$ and $\boldsymbol{w_t}$ will be denoted as u_i and $w_{t,i}$ respectively. The inner product of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ will be denoted as $\boldsymbol{u}^\top \boldsymbol{v}$ and is a scalar defined as

$$\boldsymbol{u}^\top \boldsymbol{v} = \sum_{i=1}^{d} u_i v_i$$

The *outer product* of two vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ will be denoted as $\boldsymbol{v}\boldsymbol{u}^\top$ and is a $d \times d$ matrix whose entry at row i and column j is defined as $v_i u_j$. The Kronecker product of $\boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathbb{R}^m$ is a vector in $\mathbb{R}^{n \cdot m}$ denoted by $\boldsymbol{v} \otimes \boldsymbol{u}$ whose $(i-1)m + j$-th entry is $v_i u_j$. In this thesis the following norms are used: the ℓ_2 norm

$$||\boldsymbol{u}||_2 = \sqrt{\boldsymbol{u}^\top \boldsymbol{u}},$$

the ℓ_1 norm

$$||\boldsymbol{u}||_1 = \sum_{i=1}^{d} |u_i|,$$

and the ℓ_∞ norm

$$||\boldsymbol{u}||_\infty = \max_i u_i.$$

When the subscript is omitted, the ℓ_2 norm will be assumed. Matrices will be denoted with bold capital letters such as $\boldsymbol{A}$. The identity matrix will be denoted by $\boldsymbol{I}$. Random variables will also be denoted by normal capital letters. The expectation of a random variable X will be denoted by $\mathbb{E}[X]$. The probability of an event ξ will be denoted by $\mathbb{P}(\xi)$. The indicator function $\mathbb{I}[p]$ takes as input a predicate p and returns 1 if p is true 0 if p is false. The natural logarithm of x will be denoted by $\log(x)$ and the base-b logarithm by $\log_b(x)$.

2.2 Online learning

An online learning algorithm operates in rounds and on each round the algorithm is presented a single *example*. The example consists of two parts, the *feature vector* and the *label*. On round t the algorithm first observes the feature vector $\boldsymbol{x}_t$ which is a vector containing all relevant measurements regarding the t-th example. Afterwards, the algorithm outputs a *hypothesis* h_t which is a function mapping x_t to a prediction $\hat{y}_t = h_t(x_t)$. One can also think of the algorithm as just making the prediction $\hat{y}_t$. Finally, the actual label y_t is observed, the algorithm incurs a loss $\ell(\hat{y}_t, y_t)$, and the next round begins. This description is quite general and encompasses many settings. Below we describe some concrete choices which are not restricted to the online setting.

2.2.1 Labels

First, depending on the *output space* $\mathcal{Y}$, i.e. the domain that labels come from the online learning algorithm may be faced with a variety of tasks. Some of the simplest tasks include *binary* classification $\mathcal{Y} = \{-1, +1\}$, *multiclass* classification $\mathcal{Y} = \{1, 2, \ldots, K\}$, and *regression* $\mathcal{Y} = \mathbb{R}$. This thesis mainly discusses binary classification.

2.2.2 Hypothesis Space

Next, a learning algorithm must constrain itself to a class of hypotheses $\mathcal{H}$ it will be considering. The reason for this restriction is that, in a very precise sense [9], learning is not possible when the class of hypotheses is too large. Some examples of typical classes of functions are neural networks and decision trees. This thesis will consider a very simple hypothesis class, namely linear models. A linear model is defined by a vector of weights $\boldsymbol{w}$. For binary classification and regression a linear model makes its prediction via

$$\hat{y}_t = \boldsymbol{w}^\top \boldsymbol{x}_t$$

while for multiclass classification the prediction is

$$\hat{y}_t = \operatorname*{argmax}_{1 \le y \le K} \boldsymbol{w}^\top \Psi(\boldsymbol{x}_t, y)$$

where $\Psi(\boldsymbol{x}, y)$ is a *feature mapping* from $\mathbb{R}^d \times \{1, \ldots, K\}$ to $\mathbb{R}^{Kd}$. The mapping used for multiclass classification is

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{e}_y \otimes \boldsymbol{x}$$

where $\boldsymbol{e}_y$ is a canonical basis vector for $\mathbb{R}^K$ with a 1 at the y-th entry and 0 elsewhere. Typically the hypothesis space is constrained by having a *parametric*

model, i.e. $\boldsymbol{w} \in \mathbb{R}^d$ throughout learning, and by placing an upper bound on the ℓ_2 or ℓ_1 norm of $\boldsymbol{w}$.

2.2.3 Loss Functions

Perhaps the most intuitive loss function for binary classification is the $0/1$ loss

$$\ell_{01}(\hat{y}, y) = \mathbb{I}[\hat{y} \neq y]$$

which indicates if $\hat{y}$ correctly predicts y. However this loss function is not very useful computationally. For example, it is NP-complete to decide if there exists a linear model that attains a cumulative $0/1$ loss less than m on a given set of examples, by a reduction to the Open Hemisphere problem [32]. A choice that has proven useful is to replace ℓ_{01} with functions that constitute convex upper bounds of ℓ_{01}. A function $f : \mathcal{X} \to \mathbb{R}$ is convex if for any two points $x_1, x_2 \in \mathcal{X}$ and for all $t \in [0, 1]$

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$

Furthermore, if f is differentiable at x then

$$f(y) \geq f(x) + (x-y)^\top \nabla f(x)$$

for all $y \in \mathcal{X}$. In other words, if f is convex it lies above all of its linear approximations. Common choices of convex loss functions include the *hinge loss*:

$$\ell_h(\hat{y}, y) = \max(0, 1 - y\hat{y})$$

and the *logistic loss*:

$$\ell_\sigma(\hat{y}, y) = \log_2(1 + \exp(-y\hat{y})). \tag{2.1}$$

In the rest of this thesis the logistic loss will be defined using the natural logarithm. Even though this is not an upper bound on $0/1$ loss, minimizing this loss

is equivalent to minimizing (2.1). For linear models we have $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$ which makes the above loss functions *convex* in the weight vector $\boldsymbol{w}$.

2.2.4 Notions of Good Performance

The goal of an online learning algorithm is to minimize the cumulative loss it will attain over the examples it will observe. Clearly, various tasks have various degrees of difficulty. Therefore it is appropriate to compare the cumulative loss of the algorithm with the cumulative loss of a benchmark. The benchmark is usually selected to be a fixed hypothesis from the hypothesis class the algorithm is searching over, in particular the one that in hindsight minimizes its cumulative loss. Note that when analyzing an online algorithm, the loss function we use to measure the performance of the algorithm may be different from the loss function we use to measure the performance of the benchmark. For example, in Chapter 3 we provide a *mistake bound* relating the number of mistakes, i.e. the cumulative $0/1$ loss, of the algorithm with the cumulative hinge loss of the benchmark. This mismatch is common in machine learning since, for many performance measures of interest, finding the the best hypothesis is NP-hard.

When the loss of the algorithm matches the loss of the benchmark we can define the *regret* of an online learning algorithm A as

$$\mathcal{R}(A) = \sum_{t=1}^{T} \ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t) - \inf_{\boldsymbol{w}} \sum_{t=1}^{T} \ell(\boldsymbol{w}^\top \boldsymbol{x}_t, y_t).$$

The reason many online learning algorithms are practically useful is that their regret grows sublinearly with the number of examples they observe. Therefore, the average regret per example goes to zero as more and more examples are observed.

Outside the context of online learning, there are other interesting perfor-

mance measures. Sometimes we are interested in the performance of the learned hypothesis on unseen data. This is meaningful when we assume that there is a distribution $\mathcal{D}$ from which examples are drawn at random and presented to the algorithm. A typical performance measure that will be used in this thesis is the *generalization error* of a hypothesis $\boldsymbol{w}$ which can be defined as $\mathbb{P}(y\boldsymbol{w}^\top \boldsymbol{x} \leq 0)$. Here, the probability is with respect to the random draw of $(\boldsymbol{x}, y)$ from the distribution $\mathcal{D}$. Since $\mathcal{D}$ is not known, in empirical evaluations we use the empirical error rate on a previously unseen test set of examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ drawn from $\mathcal{D}$:

$$\mathrm{err}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[y_i \boldsymbol{w}^\top \boldsymbol{x}_i \leq 0].$$

In other words, the signs of the predictions $\boldsymbol{w}^\top \boldsymbol{x}_i$ are compared with the respective labels y_i and the number of disagreements is counted. Sometimes in this thesis we will report the *accuracy* of $\boldsymbol{w}$ which is defined as $1 - \mathrm{err}(\boldsymbol{w})$.

2.3 Some Online Learning Algorithms

In the literature, many online learning algorithms have been introduced [58, 45, 16, 43]. Here, we describe three that are relevant to this thesis either because we directly build on top of them as in Chapter 3 and Chapter 4 or because prior work has modified them to partially solve tasks similar to those addressed in Chapter 5. All of the algorithms are simple to implement and require time that is proportional to the size of the dataset they are processing.

2.3.1 Perceptron

Algorithm 1 describes the Perceptron, which is probably the oldest online learning algorithm [58]. For this algorithm it is possible to obtain the following mis-

Algorithm 1 Perceptron

function PERCEPTRON(T)
  $w_1 \leftarrow \mathbf{0}$
  for $t = 1, 2, \ldots, T$ **do**
    Observe x_t
    $\hat{y}_t \leftarrow \text{sign}(w_t^\top x_t)$
    Observe y_t
    if $y_t \neq \hat{y}_t$ **then**
      $w_{t+1} = w_t + y_t x_t$
    else
      $w_{t+1} = w_t$
    end if
  end for
  return $\boldsymbol{w}_{T+1}$
end function

take bound [13, Theorem 12.1]

Theorem 1. *If the Perceptron algorithm is run on a sequence* $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_T, y_T)$, *then for all* $\boldsymbol{u} \in \mathbb{R}^d$

$$\sum_{t=1}^{T} \mathbb{I}[\hat{y}_t \neq y_t] \leq L(\boldsymbol{u}) + X^2 ||\boldsymbol{u}||^2 + X||\boldsymbol{u}||\sqrt{L(\boldsymbol{u})},$$

where $X = \max_t ||\boldsymbol{x}_t||$ *and* $L(\boldsymbol{u}) = \sum_{t=1}^{T} \max(0, 1 - y_t \boldsymbol{u}^\top \boldsymbol{x}_t)$.

Theorem 1 holds for all vectors $\boldsymbol{u}$ and in particular for the one that minimizes the bound of the theorem. Therefore it is convenient to think of $\boldsymbol{u}$ as an optimal classifier. The general strategy for proving bounds for online learning algorithms is to define some measure of progress $\Phi(\boldsymbol{w}_t)$ towards $\boldsymbol{u}$ and show that with each mistake the algorithm makes, $\boldsymbol{w}_t$ comes closer to $\boldsymbol{u}$. This measure of progress is called potential function in other contexts. The classic measure of progress that was used to prove the first mistake bound for Perceptron [54] used the inner product between $\boldsymbol{w}_t$ and $\boldsymbol{u}$. The result of Theorem 1, as well as that of Theorem 2, is shown using convex duality. In this view, the Perceptron, as well as other online learning algorithms, is a simple algorithm for solving a

certain optimization problem. The measure of progress is the objective value of the dual of this problem. In any case, the proof proceeds by defining Δ_t to be the difference between the values of the potential function before and after the t-th mistake. It then shows a simple upper bound for the telescoping sum of Δ_t. Next it is shown that every time there is a mistake the potential increases by a certain amount. Putting these pieces together implies the mistake bound. For more details the reader is referred to [61].

2.3.2 Winnow and Multiplicative Updates

When the optimal weight vector in the hypothesis space $\mathcal{H}$ is *sparse*, i.e. most of its elements are zero, then it is possible to design an online learning algorithm that makes fewer mistakes both in theory and in practice. This is one of the reasons such an algorithm will be employed in Chapter 3. The main idea is to change the prediction rule while keeping the update the same as the Perceptron. We now distinguish the *parameters* of the algorithm, $\boldsymbol{\theta}_t$ which are used for updating from its *weights* $\boldsymbol{w}_t$ which are used for prediction. The algorithm we will consider here is called Winnow [52] and connects the weights and the parameters via the following relationship

$$w_{t,i} = \frac{\exp(\theta_{t,i})}{\sum_j \exp(\theta_{t,j})}.$$

Pseudocode for Winnow is shown in Algorithm 2. Note that the decision of the algorithm is a convex combination of its features. If all the features are positive, then the predictions of the algorithm will always be positive. A simple fix that ensures that the algorithm can give both positive and negative predictions is to use an extended feature mapping

$$\Psi(\boldsymbol{x}) = [1, -1] \otimes \boldsymbol{x},$$

Algorithm 2 Balanced Winnow Algorithm

function BALANCED-WINNOW(T,α)
  $\boldsymbol{\theta}_1 \leftarrow \mathbf{0}$
  for $t = 1, 2, \ldots, T$ **do**
    Observe $\boldsymbol{x}_t$
    $w_{t,i} \leftarrow \dfrac{\exp(\theta_{t,i})}{\sum_{j=1}^{d} \exp(\theta_{t,j})}$
    $\hat{y}_t \leftarrow \operatorname{sign}(\boldsymbol{w}_t^\top \boldsymbol{x}_t)$
    Observe y_t
    if $y_t \neq \hat{y}_t$ **then**
      $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha y_t \boldsymbol{x}_t$
    else
      $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$
    end if
  end for
  return $\boldsymbol{\theta}_{T+1}$
end function

and the algorithm is referred to as Balanced Winnow.

Winnow can be implemented without the use of the parameters, by simply updating the weights multiplicatively via

$$w_{t+1,i} = \frac{w_{t,i} \exp(\alpha y x_{t,i})}{\sum_j w_{t,j} \exp(\alpha y x_{t,j})}.$$

Therefore Winnow is sometimes referred to as a multiplicative update algorithm. Nevertheless, we will stick to the formulation with the parameters $\boldsymbol{\theta}$ because those, together with the feature mapping $\Psi(\boldsymbol{x})$ will enable us to reduce the memory requirements for Balanced Winnow. We defer this discussion to Section 3.2.1.

For balanced Winnow the following mistake bound holds [13, Theorem 12.2]

Theorem 2. *If Balanced Winnow is run on a sequence* $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_T, y_T)$ *with* $\alpha = 2\epsilon/X_\infty^2$*, then for all* $0 < \epsilon < 1$ *and all* $\boldsymbol{u}$ *in the* d*-dimensional simplex*

$$\sum_{t=1}^{T} \mathbb{I}[\hat{y}_t \neq y_t] \leq \frac{1}{1-\epsilon} L(\boldsymbol{u}) + \frac{X_\infty^2 \log d}{2\epsilon(1-\epsilon)}$$

where $X_\infty = \max_t ||\boldsymbol{x}_t||_\infty$ *and* $L(\boldsymbol{u}) = \sum_{t=1}^{T} \max(0, 1 - y_t \boldsymbol{u}^\top \boldsymbol{x}_t)$.

It is also possible to obtain a bound whose form is similar to the mistake bound of Perceptron:

$$\sum_{t=1}^{T} \mathbb{I}[\hat{y}_t \neq y_t] \leq L(\boldsymbol{u}) + 4X_\infty^2 \log d + 2X_\infty \sqrt{L(\boldsymbol{u}) \log d}$$

by varying the learning rate α on each round, depending on the number of mistakes so far [61, Corollary 6].

Finally, note that since $\boldsymbol{w}_t$ is always inside the d-dimensional probability simplex, the theorem assumes the same is true for all $\boldsymbol{u}$. Since $\boldsymbol{w}_t$ and $\boldsymbol{u}$ can be viewed as probability distributions, a natural measure of progress is the relative entropy between $\boldsymbol{w}_t$ and $\boldsymbol{u}$:

$$\Phi(\boldsymbol{w}_t) = D(\boldsymbol{u}||\boldsymbol{w}_t) = \sum_{i=1}^{d} u_i \log \frac{u_i}{w_{t,i}}$$

which is always nonnegative. The same measure of progress will be used in the proof of Theorem 4 in Chapter 3.

2.3.3 Online Gradient Descent

The last algorithm that we will review is Online Gradient Descent (OGD). OGD is described in Algorithm 3 for the specific task of finding a minimizer of the cumulative loss over a set of examples. The loss functions are assumed to be convex and Lipschitz. In line 7 of the algorithm the notation $\nabla_w \ell(p_t, y_t)$ means the gradient of the loss with respect to w evaluated at the prediction p_t and label y_t. Furthermore, the hypothesis space is all the vectors that belong to a closed convex set A. Usually, the Euclidean projection in line 8 is assumed to be easy computationally. For example, this is true when $A = \{v \in \mathbb{R}^d : ||v|| \leq R\}$, i.e. A is an ℓ_2 ball of radius R.

Online gradient descent is a robust algorithm that works in a much broader context than described here. For example, it can be shown that it can also work

Algorithm 3 Online Gradient Descent. Solves $\boldsymbol{w} = \text{argmin}_{w \in A} \sum_t \ell(\boldsymbol{w}^\top \boldsymbol{x}_t, y_t)$

function OGD(η)
  $\boldsymbol{w}_1 \leftarrow \mathbf{0}$
  for $t = 1, 2, \ldots, T$ **do**
    Observe $\boldsymbol{x}_t$
    Predict $p_t \leftarrow \boldsymbol{w}_t^\top \boldsymbol{x}_t$
    $\eta_t = \eta / \sqrt{t}$
    $\boldsymbol{w}'_t \leftarrow \boldsymbol{w}_t - \eta_t \nabla_w \ell(p_t, y_t)$
    $\boldsymbol{w}_{t+1} \leftarrow \text{argmin}_{w \in A} ||\boldsymbol{w} - \boldsymbol{w}'_t||^2$
  end for
  return $\bar{\boldsymbol{w}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}_t$
end function

for the task of online convex optimization where the loss functions may be changing from round to round [67]. It also works with high probability when instead of having access to the gradient, the algorithm only observes a noisy version of it [63]. Online Gradient Descent entertains the following regret bound whose simple proof is included:

Theorem 3. *Assume that ℓ is convex and that $||\nabla_w \ell(p, y)||^2 \leq G$. Let D be the diameter of the set A i.e. $\sup\{||w - u|| : w, u \in A\} \leq D$. Then for every $\boldsymbol{u} \in A$, the sequence of vectors $\boldsymbol{w}_t$ produced by Online Gradient Descent satisfy*

$$\sum_{t=1}^{T} \ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t) - \ell(\boldsymbol{u}^\top \boldsymbol{x}_t, y_t) \leq (G\eta + \frac{D^2}{2\eta})\sqrt{T}$$

Proof. As a shorthand we will denote $\ell(\boldsymbol{v}^\top \boldsymbol{x}_t, y_t)$ by $\ell_t(\boldsymbol{v})$ and $||\nabla_w \ell(p_t, y_t)||^2$ by g_t. From the convexity of ℓ we have that

$$\eta_t(\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{u})) \leq \eta_t(\boldsymbol{w}_t - \boldsymbol{u})^\top \boldsymbol{g}_t.$$

Adding and subtracting $\frac{1}{2}||\boldsymbol{w}_t - \boldsymbol{u}||^2 + \eta_t^2/2||\boldsymbol{g}_t||^2$ leads to

$$\begin{aligned}
\eta_t(\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{u})) &\leq \frac{1}{2}||\boldsymbol{w}_t - \boldsymbol{u}||^2 + \frac{\eta_t^2}{2}||\boldsymbol{g}_t||^2 - \frac{1}{2}||w_t - u - \eta_t \boldsymbol{g}_t||^2 \\
&\leq \frac{1}{2}||\boldsymbol{w}_t - \boldsymbol{u}||^2 + \frac{\eta_t^2}{2}G - \frac{1}{2}||\boldsymbol{w}'_t - \boldsymbol{u}||^2 \\
&\leq \frac{1}{2}||\boldsymbol{w}_t - \boldsymbol{u}||^2 + \frac{\eta_t^2}{2}G - \frac{1}{2}||\boldsymbol{w}_{t+1} - \boldsymbol{u}||^2
\end{aligned}$$

where the last inequality follows from the fact that w'_t is further from every point inside A than its projection w_{t+1}. Dividing both sides by η_t leads to

$$\ell(\boldsymbol{w}_t) - \ell(\boldsymbol{u}) \leq \frac{||\boldsymbol{w}_t - \boldsymbol{u}||^2 - ||\boldsymbol{w}_{t+1} - \boldsymbol{u}||^2}{2\eta_t} + \frac{\eta_t}{2} G. \tag{2.2}$$

We now simply sum (2.2) over t and obtain

$$\begin{aligned}
\sum_{t=1}^{T} (\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{u})) &\leq \frac{G}{2} \sum_{t=1}^{T} \eta_t + \frac{||\boldsymbol{w}_1 - \boldsymbol{u}||^2}{2\eta_1} - \frac{||\boldsymbol{w}_{T+1} - \boldsymbol{u}||^2}{2\eta_T} \\
&+ \sum_{t=2}^{T} \left(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}} \right) ||\boldsymbol{w}_t - \boldsymbol{u}||^2 \\
&\leq \frac{G}{2} \sum_{t=1}^{T} \eta_t + D^2 \left(\frac{1}{2\eta_1} + \sum_{t=2}^{T} \left(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}} \right) \right),
\end{aligned}$$

where in the last inequality we used that $||w_t - u|| \leq D^2$ for all t and dropped the non-positive term $-\frac{||w_{T+1}-u||^2}{2\eta_T}$. Now the telescoping sum in the right hand side cancels out leaving

$$\sum_{t=1}^{T} (\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{u})) \leq \frac{G}{2} \sum_{t=1}^{T} \eta_t + \frac{D^2}{2\eta_T}$$

Substituting $\eta_t = \eta/\sqrt{t}$ we get

$$\sum_{t=1}^{T} (\ell_t(w_t) - \ell_t(u)) \leq \frac{G\eta}{2} \sum_{t=1}^{T} \frac{1}{\sqrt{t}} + \frac{D^2}{2\eta} \sqrt{T} \leq \frac{G\eta}{2} \int_0^T \frac{dt}{\sqrt{t}} + \frac{D^2}{2\eta} \sqrt{T} = (G\eta + \frac{D^2}{2\eta}) \sqrt{T}$$

□

2.4 Active Learning

In active learning, a learner is given access to unlabeled data and is allowed to adaptively choose which ones to label. This learning model is motivated by applications in which the cost of labeling data is high relative to that of collecting

the unlabeled data itself, as labeling often requires manual intervention, experimentation, or evaluation. Therefore, the hope is that the active learner only needs to query the labels of a small number of the unlabeled data, and otherwise perform as well as a fully supervised learner. This thesis, looks into *agnostic active learning algorithms*, which work under the same assumptions as supervised learning. Though this is in contrast with earlier work which assumed a noise-free setting [15, 20] some of the basic ideas remain the same.

In the noise-free setting [15, 20], the algorithm basically maintains a candidate set of hypotheses, a so called *version space*, and asks for the label of an example only if there is disagreement within this set about how to label the example. In the noisy setting the version space is defined as the set of hypotheses that cannot be distinguished from the empirically best via a confidence bound [2, 21, 4, 36, 46]. As more labels are collected this set can quickly shrink by removing all the hypotheses that with high probability are provably worse than the empirically best hypothesis.

The version space approach unfortunately has its share of significant drawbacks. The first is computational intractability: maintaining a version space and guaranteeing that *only* hypotheses from this set are returned is difficult for linear predictors and appears intractable for interesting nonlinear predictors such as neural nets and decision trees [15]. Secondly, maintaining a version space can be brittle: a single mishap, due to, say, modeling failures or computational approximations, might cause the learner to exclude the best hypothesis from the version space forever; this is an ungraceful failure mode that is not easy to correct. A third drawback, shared by many active learning algorithms including the one in the next section, is related to sample re-usability: if labeled data is collected using a version space-based active learning algorithm, and one later

Algorithm 4 Label Efficient Perceptron

function LEP(c,T)
  $\boldsymbol{w}_1 \leftarrow \mathbf{0}$
  for $t = 1, 2, \ldots, T$ **do**
    Observe x_t
    $p_t \leftarrow \boldsymbol{w}_t^\top \boldsymbol{x}_t$
    $\hat{y}_t \leftarrow \text{sign}(p_t)$
    Draw a Bernoulli random variable Z_t with $\mathbb{P}(Z_t = 1) = c/(c + |p_t|)$
    if $Z_t = 1$ **then**
      Query the oracle for y_t
      $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t \mathbb{I}[y_t \neq \hat{y}_t] \boldsymbol{x}_t$
    else
      $w_{t+1} = w_t$
    end if
  end for
  return $\boldsymbol{w}_{T+1}$
end function

decides to use a different algorithm or hypothesis class, then the earlier data may not be freely re-used because its collection process is inherently biased.

In the next section the Label Efficient Perceptron algorithm is first introduced. This algorithm achieves in expectation the same mistake bound as the Perceptron. It does not maintain a version space, and is simple to implement. It is analyzed in a worst case setting and uses randomization to avoid being fooled by an adversary. Even though there is no quantification on the reduction in amount of requested labels, this algorithm is a first step towards theoretically sound active learning.

2.4.1 Label Efficient Perceptron

The Label Efficient Perceptron (LEP) algorithm is shown in Algorithm 4. For this algorithm, it can be shown that its expected number of mistakes is not much larger than the cumulative hinge loss $L(\boldsymbol{u}) = \sum_{t=1}^{T} \max(0, 1 - y_t \boldsymbol{u}^\top x_t)$ of any vector $\boldsymbol{u} \in \mathbb{R}^d$ over the sequence $(x_1, y_1), (x_2, y_2), \ldots (x_T, y_T)$. In particular [13,

Theorem 12.5] proves that

$$\mathbb{E}\left[\sum_{t=1}^{T}\mathbb{I}[\hat{y}_t \neq y_t]\right] \leq \left(1+\frac{X^2}{2c}\right)L + \frac{||\boldsymbol{u}||^2(2c+X^2)^2}{8c} \tag{2.3}$$

where $X = \max_t ||x_t||$.

Even though this result is obtained by a worst case analysis, the statistical setting, described in Section 2.4.2, shares some of the same ideas. The first difference between Perceptron and LEP is the introduction of randomization. It is not hard to see that if the algorithm had adopted a deterministic policy for which labels to ask, then an adversary could have manipulated the information the algorithm collects. The second difference is that $|p_t|$ plays a crucial role in the algorithm. Notice that $|p_t|$ is a proxy for the algorithm's confidence in its prediction. A large $|p_t|$ means that if the algorithm is wrong it will suffer a large hinge loss on round t. This in turn can be related to the observation that, in the proof of the Perceptron, a mistake with large loss leads to an update that makes a lot of progress towards a good model. Querying the oracle for y_t with probability $c/(c+|p_t|)$ is thus a simple way for the algorithm to hedge its bets.

2.4.2 Importance Weighted Active Learning

The label efficient Perceptron is a good first step to understand some of the issues in active learning. However the analysis is far from satisfactory. First, the guarantee it provides is a bound on the expected number of mistakes. However, one would prefer to have a high probability bound relating the generalization error of the actively learned classifier with that of a classifier trained on all the examples. Second, the label efficient Perceptron does not come with a bound on the number of requested labels. Third it is not clear if the labeled sample collected from a run of the algorithm is biased or it could be reused with another

algorithm.

Importance Weighted Active Learning (IWAL) [4, 5] addresses these problems and is much more generic in the sense that it is a *reduction* from active learning to supervised learning. This allows to use learning algorithms other than linear classifiers although this thesis does not explore this possibility. Furthermore, the analysis in [5] provides a bound on the number of requested labels. Moreover, IWAL is provably *consistent*, i.e. given infinite data it converges to an optimal hypothesis in a given hypothesis class. Finally, labeled data collected with IWAL can be re-used with other algorithms.

To sidestep all of the above problems, [5] instantiate the framework of [4] using a rejection threshold similar to the algorithm of [21]. Unlike [21] however hypotheses are accessed via a much simpler oracle which matches an abstraction of many supervised learning algorithms. Hence active learning algorithms built in this way are immediately and widely applicable.

We defer the description of the algorithm of [5] to Chapter 5 of this thesis alongside with the details of a practical implementation for the case of linear predictors. The final algorithm is as efficient as the label efficient Perceptron but works with many loss functions, and returns a reusable labeled dataset. This is in sharp contrast to many previous algorithms such as [15, 2, 3, 21, 36, 46] that create heavily biased data sets. Even though some approximations are made in order to attain a simple algorithm, these do not affect the consistency of the algorithm.

2.5 Prediction Suffix Trees

Prediction Suffix Trees (PSTs), also known as Context Trees, are a well studied and compact model for problems such as temporal classification and probabilis-

tic modeling of sequences [11, 37, 55, 57, 65]. Different variants of PSTs are also called context tree weighting [65] and variable length Markov Models [11]. PSTs operate in a setting similar to online learning. The model observes symbols from a sequence, one at a time, and makes a prediction about the next symbol based on the symbols it has observed so far. PSTs typically use only a few recently observed symbols which are called the context of the prediction. In this sense they are making a Markovian assumption but, unlike most other Markov models, the number of symbols that are used to predict the next symbol depends on the specific context in which the prediction is made. PSTs will be described in greater detail in Chapter 3. The rest of this section provides a high level overview of PSTs.

As the name suggests, a PST is a tree whose nodes correspond to suffixes of the sequence observed so far. If the sequence comes from an alphabet Σ, then each node in the PST has up to $|\Sigma|$ children. The root of the tree corresponds to the empty suffix. The children of the root correspond to suffixes of length 1, their children correspond to suffixes of length 2 and so on. The edge from a parent to a child is labelled with a symbol from the alphabet. If the parent corresponds to suffix $s \in \Sigma^k$ and the child corresponds to suffix $rs \in \Sigma^{k+1}$, then the edge is labelled by $r \in \Sigma$.

Given a sequence $y_1, y_2, \ldots, y_{t-1}$, a Markov model of order k can be viewed as a full tree of depth k where estimates of the probabilities of the form $p(y_t|y_{t-1}, \ldots, y_{t-k})$ are stored in the leaves of the tree. In other words, each instantiation of the variables upon which the model is conditioning, is a path in this tree. Furthermore, this path goes through the nodes that correspond to the suffixes of the sequence.

Prediction Suffix Trees generalize Markov models by allowing the tree to

take any shape. This relaxation matches the properties of many natural sequential problems, such as language modelling, where the right amount of context depends on the context itself. For example, in English, it is safe to predict that a 'u' will appear after a 'q' while it is less clear which letter will appear after the sequence 'en'. A PST for modelling English can therefore allocate a small subtree to handle suffixes ending with 'q' and a larger tree to handle more uncertain suffixes. In other words, a PST can model long range dependencies without any blow-up in the number of parameters used by being frugal when a complex model is not necessary.

So far, the discussion has only treated the leaves as the parts of the tree that carry all the information about the model. However, internal nodes can also be used to store estimates of lower order Markov models. For example, the first level of nodes in the tree can store estimates of $p(y_t|y_{t-1})$, a so-called bigram model. The models stored in the upper levels of the tree are simple but can be estimated reliably. The models stored in the lower levels are more flexible but their reliable estimation requires large amounts of data. A simple way to sidestep the dilemma of which model to use is to use all the models by taking a weighted average of the recommendations of each model. In our adaptation of PSTs in Chapter 3, we will use a similar weighting of all the nodes in the tree.

CHAPTER 3

SEQUENTIAL PREDICTION WITH A SMALL TREE

3.1 Introduction

This chapter shows how to learn a small and accurate Prediction Suffix Tree (PST) for sequential prediction by a modified multiplicative update algorithm. The reason we employ multiplicative updates is rather involved, but it will become clearer during the analysis of the algorithm. Roughly speaking, in order to maintain a small tree, the performed updates are approximate and not as effective as the ones that would be used if no constraint on the size of the tree were imposed. This ineffectiveness can be quantified by the norm of a specific vector, and this norm is minimized when the updates are multiplicative.

In [23] the Perceptron algorithm is used for sequential prediction. Their theoretical results are similar but inferior in the sense that the upper bound on the size of the PST here is tighter. The mistake bounds are incomparable, and it is well known that additive and multiplicative updates have different merits. The experiments of this chapter verify empirically that the proposed algorithm consistently obtains lower error rates and grows smaller trees than the algorithm of [23].

The motivating application is monitoring processes in a computer system. Each process produces a sequence of system calls that request different services from the operating system. The task is to model this sequence and maintain a compact profile of the application. In this setting, it is expected that a multiplicative update algorithm is more appropriate than an perceptron-like additive algorithm for two reasons. First, complex applications are likely to exhibit different behaviors at various points during their execution, and this is actually

observed in our experiments. A multiplicative algorithm can quickly modify its model to reflect this. Second, multiplicative algorithms cope better with many irrelevant attributes, and this will turn out to be true when the problem is formalized.

3.2 Learning Prediction Suffix Trees

As noted in Section 2.5, the advantage of PSTs compared to other Markov models is that the number of symbols used to predict the next symbol depends on the context in which the prediction is made. But how much context should be used for each prediction? When learning a PST, one is faced with the following dilemma. If one considers contexts with more symbols than necessary, one must estimate more parameters and this will likely hurt the performance of the algorithm. On the other hand, underestimating the optimal context will typically lead to many mistakes.

The goal is to come up with an algorithm that learns a PST of appropriate size by balancing this tradeoff. The algorithm in [23] uses the perceptron algorithm to learn a PST. The basic idea is that choosing not to grow the tree is like introducing a deterministic noise term in the perceptron update. The same idea is used here for Balanced Winnow, but many details are different, especially how much noise is tolerated before the tree is grown. Before showing how Balanced Winnow can be adapted to work for PSTs, a digression is necessary to show how to modify Balanced Winnow so that it effectively ignores some of its inputs.

3.2.1 Sparse Balanced Winnow

Algorithm 2 on page 14 assigns a nonzero weight to every feature, even if some of these weights may be exponentially small. Here, an equivalent formulation is given so that at prediction time some attributes do not have to be computed. Viewed as a linear classifier, Balanced Winnow has assigned zero weight to those attributes. When many of the attributes have a zero weight the resulting hypothesis is sparse. This may at first seem hard given that the elements of the weight vector are by construction always positive and that the proof of Theorem 2 hinges on the use of relative entropy which becomes infinite if one of the entries of the weight vector is zero. However, if one considers the form of the input $\boldsymbol{x}_t = [1, -1] \otimes \boldsymbol{x}_t^+$ and noticing that

$$\boldsymbol{w}_t^\top \boldsymbol{x}_t = \boldsymbol{w}_t^{+\top} \boldsymbol{x}_t^+ - \boldsymbol{w}_t^{-\top} \boldsymbol{x}_t^+ = (\boldsymbol{w}_t^+ - \boldsymbol{w}_t^-)^\top \boldsymbol{x}_t^+,$$

where $\boldsymbol{w}_t^+$ corresponds to the first half of the weight vector $\boldsymbol{w}_t$ and $\boldsymbol{w}_t^-$ corresponds to its second half, then if $w_{t,i}^+ = w_{t,i}^-$ any decision of the form $\boldsymbol{w}_t^\top \boldsymbol{x}_t$ will effectively ignore attribute $x_{t,i}$. A better insight on how the algorithm operates with inputs of the form $[1, -1] \otimes \boldsymbol{x}_t^+$ is provided if the parameters $\boldsymbol{\theta}_t$ are rewritten in the same manner as the weight vector, i.e. $\boldsymbol{\theta}_t = [\boldsymbol{\theta}_t^+, \boldsymbol{\theta}_t^-]$. The following lemma shows the relationship between the two vectors $\boldsymbol{\theta}_t^+$ and $\boldsymbol{\theta}_t^-$

Lemma 1. *For all t, Balanced Winnow maintains $\boldsymbol{\theta}_t^- = -\boldsymbol{\theta}_t^+$.*

Proof. The proof is by induction. For $t = 1$ we have $\boldsymbol{\theta}_1 = \mathbf{0}$ so $\boldsymbol{\theta}_t^+ = \boldsymbol{\theta}_t^- = \mathbf{0}$ and the claim is true. If at time $t + 1$ there was no mistake, then the claim is true. If there was a mistake, then the parameters were updated as follows: $\boldsymbol{\theta}_{t+1}^+ = \boldsymbol{\theta}_t^+ + \alpha y_t \boldsymbol{x}_t^+$ and $\boldsymbol{\theta}_{t+1}^- = \boldsymbol{\theta}_t^- + \alpha y_t (-\boldsymbol{x}_t^+)$. Using the induction hypothesis, the latter update is $\boldsymbol{\theta}_{t+1}^- = -\boldsymbol{\theta}_t^+ - \alpha y_t \boldsymbol{x}_t^+ = -\boldsymbol{\theta}_{t+1}^+$ □

Since $w_{t,i} = \dfrac{e^{\theta_{t,i}}}{\sum_{j=1}^{d} e^{\theta_{t,i}}}$, one can only have $w_{t,i}^{+} = w_{t,i}^{-}$ when $\theta_{t,i}^{+} = \theta_{t,i}^{-}$. Using the lemma above one concludes that it must be $\theta_{t,i}^{-} = \theta_{t,i}^{+} = 0$ to make sure that $\boldsymbol{w}_t^{\top}\boldsymbol{x}_t$ does not depend on the value of $x_{t,i}$.

Another way to see this is to note that by Lemma 1, the decision at time t can be written as:

$$\begin{aligned}\hat{y}_t &= = (\boldsymbol{w}_t^{+} - \boldsymbol{w}_t^{-})^{\top}\boldsymbol{x}_t^{+} = \sum_{i=1}^{d} \frac{e^{\theta_{t,i}^{+}} - e^{\theta_{t,i}^{-}}}{\sum_{j=1}^{d} e^{\theta_{t,j}^{+}} + e^{\theta_{t,j}^{-}}} x_{t,i}^{+} \\ &= \sum_{i=1}^{d} \frac{\sinh(\theta_{t,i}^{+})}{\sum_{j=1}^{d} \cosh(\theta_{t,j}^{+})} x_{t,i}^{+},\end{aligned}$$

where

$$\begin{aligned}\sinh(\theta) &= \frac{e^{\theta} - e^{-\theta}}{2} \\ \cosh(\theta) &= \frac{e^{\theta} + e^{-\theta}}{2}.\end{aligned}$$

For the purposes of decisions, the quantity $\sum_{j=1}^{d} \cosh(\theta_{t,j}^{+})$ is a positive constant so it will not affect the sign of $\hat{y}_t$. Let $g_{t,i} = \sinh(\theta_{t,i}^{+})$ be the unnormalized difference of the weights $w_{t,i}^{+}$ and $w_{t,i}^{-}$. The value of $x_{t,i}^{+}$ is ignored when $g_{t,i} = 0$ which is true if and only if $\theta_{t,i}^{+} = 0$.

The strategy to derive a multiplicative update algorithm whose hypothesis is sparse is to implement the algorithm in the space of $\boldsymbol{\theta}$ parameters. To make sure that some of the parameters remain to zero, we will introduce deterministic noise that cancels the update for these specific parameters. The analysis of the algorithm will be done in the space of weights $\boldsymbol{w}$. When analyzing the algorithm, we will have to show that the progress made with each mistake overshadows the effect of noise. The algorithm will be phrased in terms of the normalized weights to maintain correspondence with the analysis and with algorithm 2 in page 14. In a practical implementation, one should update the values

of the parameters $\boldsymbol{\theta}_t^+$ and use the unnormalized weights $g_{t,i} = \sinh(\theta_{t,i}^+)$ to make predictions.

3.2.2 Winnow for Prediction Suffix Trees

The formal setup is as follows. The task is to predict the items of a sequence $y_1, y_2, \ldots, y_T$, $y_i \in \Sigma$, one at a time. For now, it is assumed that $\Sigma = \{-1, 1\}$; extensions to larger alphabets are discussed in Section 3.3. Let $|y|$ be the length of a sequence $y \in \Sigma^*$, where Σ^* is the Kleene closure of Σ, and let y_i^j denote the subsequence $y_i, \ldots, y_j$. The prediction for y_t will be based on a suffix of y_1^{t-1}, typically one that contains only the last few symbols in y_1^{t-1}. A good initial assumption is that symbols near t should be more important for the prediction task than symbols away from t [37, 65]. This can be encoded in the features, which will be of the form $\boldsymbol{x}_t = [1, -1] \otimes \boldsymbol{x}_t^+$, where:

$$x_{t,s}^+ = \begin{cases} \beta^{|s|} & \text{if} \quad s = y_{t-i}^{t-1} \qquad i = 1, \ldots, t-1 \\ 0 & \text{otherwise} \end{cases}$$

and $0 < \beta < 1$ will be specified later. Therefore the dimensionality of $\boldsymbol{x}_t$ will be twice the number of distinct substrings of y_1^T with a trivial upper bound of $T(T-1)$. Here and below, the vector $\boldsymbol{x}^+$ is indexed by strings from the language $\Sigma^{\leq T}$ and the notation $x_{t,s}^+$ simply means $x_{t,\ell(s)}^+$, where $\ell(s)$ is the position of s in the lexicographic order of $\mathcal{Y}_T$, the set of all substrings of y_1^T. Another useful function is the inverse of $\ell(\cdot)$: $s(i)$ is the string at position i in the aforementioned lexicographic order. Furthermore, we extend $s(\cdot)$ to be a periodic function with period $|\mathcal{Y}_T|$. This allows to handle indices from the extended features $\boldsymbol{x}_t$, so that we can write $|x_{t,i}| = x_{t,s(i)}^+$ for example.

As before, the algorithm will have to learn a vector of parameters $\boldsymbol{\theta}_t = [\boldsymbol{\theta}_t^+, \boldsymbol{\theta}_t^-]$ while keeping its *support*, the strings for which the corresponding en-

tries are nonzero, small. The decisions of the algorithm will be of the form:

$$\hat{y}_t = \sum_i g_{t,i} x_{t,i}^+$$

where $g_{t,i} = \sinh(\theta_{t,i}^+)$ and i ranges over the support of $\boldsymbol{\theta}_t$. The set A that contains the support of $\boldsymbol{\theta}$, every suffix of every element in the support, and the empty string can be viewed as a tree. This tree is a Prediction Suffix Tree (PST) and is constructed in the following way. Every element of A is a node in the tree. The root corresponds to the empty string. Let $y \in \Sigma$ and $u \in \Sigma^*$. If $v = yu \in A$, then $u \in A$ by the definition of A and v is a direct child of u in the tree with the link from u to v being labeled with y. In this view, $\boldsymbol{\theta}^+$, assigns weights to nodes in the tree, and the predictions of the online algorithm amount to taking a weighted sum of the values in the nodes of a path in the tree. This path starts from the root and follows the links that correspond to the symbols y_{t-1},$y_{t-2}, \ldots$, until reaching a leaf. Figure 3.1 shows a PST. In this figure, if $\beta = 3/4$ and $y_{t-2}^t = -1 - 1 + 1$ then the prediction for symbol y_{t+1} will be

$$\hat{y}_{t+1} = \text{sign}\left(\frac{3}{4}\sinh(-2) + \frac{9}{16}\sinh(-\frac{1}{2}) + \frac{27}{64}\sinh(3)\right) = +1.$$

It is thus possible for the algorithm to overcome the prior belief that a longer suffix is less relevant by assigning its node a large value.

To use Balanced Winnow for learning a PST, note that the algorithm starts with $\boldsymbol{\theta}_1 = \mathbf{0}$, therefore initially the support of $\boldsymbol{\theta}$ is the empty string and the corresponding tree has only one node. As learning progresses, each mistake the algorithm makes causes some of the parameters to be updated to nonzero values. Hence the support of $\boldsymbol{\theta}$ is expected to increase. In fact, the algorithm requires that the support of $\boldsymbol{\theta}_t$ be a subset of the support of $\boldsymbol{\theta}_{t+1}$. Thus, the algorithm can be viewed as growing the tree that corresponds to A. Even if an update causes a parameter to become zero after it has taken a nonzero value, the algorithm will still keep the corresponding node in the tree.

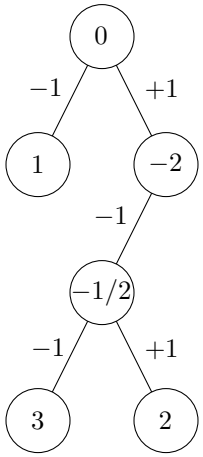

Figure 3.1: A PST whose support is the union of -1,$+1-1-1, +1-1+1$ and all their suffixes.

If we only apply Balanced Winnow to the task of learning such a PST it will need memory that can be as large as $O(T^2)$. A random sequence of $\{-1, +1\}$ symbols can achieve this. This happens because every distinct substring of y_1^T will have an associated parameter; put otherwise, at round t there will be $O(t)$ new parameters to adapt, each corresponding to the $O(t)$ new suffixes of y_1^t. The algorithm's strategy will therefore be to have an adaptive bound d_t on the length of the suffixes that it will consider, which is the same as the depth up to which it will grow the tree on round t.

The proposed algorithm modifies Balanced Winnow so that the parameter update is

$$\theta_{t+1,i} = \begin{cases} \theta_{t,i} + \alpha y_t x_{t,i} & \text{if } |s(i)| \leq d_t \\ \theta_{t,i} & \text{otherwise.} \end{cases}$$

This can be equivalently written as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha y_t \boldsymbol{x}_t + \alpha \boldsymbol{n}_t,$$

where $\boldsymbol{n}_t$ is a deterministic noise vector that cancels some of the components of

the update:

$$n_{t,i} = \begin{cases} -y_t x_{t,i} & \text{if } |s(i)| > d_t \\ 0 & \text{otherwise.} \end{cases}$$

Note that $||\boldsymbol{n}_t||_\infty = \beta^{(d_t+1)}$, a fact that will be used later in the derivation of a mistake bound. Let h_t be the length of the path from the root to a leaf using the symbols $y_{t-1}, y_{t-2}, \ldots$. Clearly, if at any point the algorithm's way of setting d_t suggests that it should be less than h_t, then there is no reason to let this happen because the tree has already grown beyond this point. Letting $d_t < h_t$ will only make the norm of $\boldsymbol{n}_t$ larger without any computational benefit. Therefore $d_t \geq h_t$ which makes it is easy to prove the following

Lemma 2. *For all t and i, either $\theta_{t,i} = 0$ or $n_{t,i} = 0$.*

Proof. There are two cases: either $|s(i)| \leq d_t$ or $|s(i)| > d_t$. If $|s(i)| \leq d_t$ then $n_{t,i} = 0$ by definition. Now suppose $\theta_{t,i} \neq 0$ and $|s(i)| > d_t$. That means there was a point $t' < t$ when $d_{t'} > d_t$. That would imply $d_{t'} > h_t$, but since the tree never shrinks this cannot happen. Therefore $\theta_{t,i} = 0$ if $|s(i)| > d_t$. □

The lemma implies that $\boldsymbol{\theta}_t^\top \boldsymbol{n}_t = 0$. Furthermore the sum of the norms of the noise vectors will turn up in the proof of the mistake bound, therefore the algorithm should keep it bounded by a function of the number of mistakes. Let J_t be the subset of rounds $1, \ldots, t$ in which a mistake was made, and let $M_t = |J_t|$. Also let

$$P_t = \sum_{i \in J_t} ||n_i||_\infty = \sum_{i \in J_t} \beta^{(d_i+1)}$$

and $M_0 = P_0 = 0$. The algorithm will require that d_t is the smallest integer such that

$$P_t \leq M_t^{2/3} \tag{3.1}$$

subject to $d_t \geq h_t$. It is also worth noting that other sublinear functions of M_t can be used to bound P_t. This specific choice empirically works very well.

Now it is possible to prove the following:

Lemma 3. *Setting*

$$d_t = \max\{h_t, \left\lceil \log_\beta(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}) - 1 \right\rceil\} \tag{3.2}$$

ensures that for all t, $P_t \leq M_t^{2/3}$.

Proof. The proof is by induction. For $t = 1$ the claim clearly holds. If there is no mistake on round t, then $P_t = P_{t-1}$ and $M_t = M_{t-1}$ so the claim holds. If there is a mistake, then $P_t = P_{t-1} + ||\boldsymbol{n}_t||_\infty$, $M_t = M_{t-1} + 1$ and it suffices to show that $P_t^3 \leq P_{t-1}^3 + 2P_{t-1}^{3/2} + 1$ since by induction $P_{t-1}^3 + 2P_{t-1}^{3/2} + 1 \leq (M_{t-1} + 1)^2$. So it must be that

$$(P_{t-1} + ||\boldsymbol{n}_t||_\infty)^3 \leq P_{t-1}^3 + 2P_{t-1}^{3/2} + 1 \tag{3.3}$$

Let $z = ||\boldsymbol{n}_t||_\infty > 0$. Inequality (3.3) can be written as

$$z^3 + 3P_{t-1}z^2 + 3P_{t-1}^2 z - 2P_{t-1}^{3/2} - 1 \leq 0$$

and is valid when z is less than its only real root:

$$z \leq \sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}$$

Since $z = ||\boldsymbol{n}_t||_\infty = \beta^{(d_t+1)}$, the proof is concluded by

$$\beta^{(d_t+1)} \leq \sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}$$

$$d_t \geq \log_\beta \left(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1} \right) - 1.$$

The statement of the lemma simply enforces that d_t is an integer and that $d_t \geq h_t$. □

Algorithm 5 Balanced Winnow for PSTs

1: **function** BW-PST(y,α,T)
2: $P_0 \leftarrow 0$
3: $A_1 \leftarrow \{\epsilon\}$ ▷ ϵ is the empty string
4: $\boldsymbol{\theta}_1 \leftarrow \mathbf{0}$
5: **for** $t = 1, 2, \ldots, T$ **do**
6: $h_t \leftarrow \max\{j : y_{t-j}^{t-1} \in A_t\}$ ▷ Depth of tree at current context
7: $d_t \leftarrow \max\{h_t, \left\lceil \log_{1-\epsilon}(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}) - 1 \right\rceil\}$ ▷ Lemma 3
8: Compute $\boldsymbol{x}_t$ from y_1^{t-1}
9: $w_{t,i} \leftarrow e^{\theta_{t,i}} / \sum_j e^{\theta_{t,j}}$
10: $\hat{y}_t \leftarrow \text{sign}(\boldsymbol{w}_t^\top \boldsymbol{x}_t)$ ▷ $\text{sign}(\hat{y}_t) = \text{sign}(\sum_i \sinh(\theta_{t,i}^+) x_{t,i}^+)$
11: **if** $\hat{y}_t \neq y_t$ **then**
12: $P_t \leftarrow P_{t-1} + (1-\epsilon)^{(d_t+1)}$
13: $A_{t+1} \leftarrow A_t \cup \{y_{t-i}^{t-1} : 1 \leq i \leq d_t\}$ ▷ Grow the tree
14: $\boldsymbol{\theta}_{t+1,i} \leftarrow \begin{cases} \boldsymbol{\theta}_{t,i} + \alpha y_t \boldsymbol{x}_{t,i} & \text{if } |s(i)| \leq d_t \\ \boldsymbol{\theta}_{t,i} & \text{otherwise} \end{cases}$
15: **else**
16: $P_t \leftarrow P_{t-1}$
17: $A_{t+1} \leftarrow A_t$
18: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$
19: **end if**
20: **end for**
21: **return** $A_{T+1}, \boldsymbol{\theta}_{T+1}$
22: **end function**

It is now easy to state how everything is put together in Algorithm 5. The set of suffixes A_t can be implemented as a tree, as described above. In practice the parameters $\theta_{t,i}^+$ should be stored and predictions should be made as

$$\hat{y}_t = \sum_i \sinh(\theta_{t,i}^+) x_{t,i}^+$$

The algorithm starts with an empty tree and in each round it predicts the next symbol using the parameters in the tree. If a mistake is made and if $d_t > h_t$ the tree is grown in line 13. Finally, the parameter values for the nodes in the tree are updated and the next symbol in the sequence is processed.

The following theorem shows how the number of mistakes of this algorithm compares against the performance of the optimal tree in hindsight. This is mea-

sured by the cumulative δ-*hinge loss* of the optimal vector $\boldsymbol{u}$. The δ-*hinge loss* that $\boldsymbol{u}$ attains at round t is defined as

$$L_t = \max\{0, \delta - y_t \boldsymbol{u}^\top \boldsymbol{x}_t\}.$$

The following two lemmas will be used in the proof of the main result.

Lemma 4. *For all $x \in [-1, 1]$ and $\alpha > 0$, $e^{\alpha x} \leq \cosh(\alpha) + \sinh(\alpha)x$.*

Proof. First note that $e^{-\alpha} = \cosh(\alpha) - \sinh(\alpha)$ and $e^{\alpha} = \cosh(\alpha) + \sinh(\alpha)$. The lemma follows by the convexity of $e^{\alpha x}$. □

Lemma 5. *For all $\alpha > 0$, $\log\cosh(\alpha) \leq \alpha^2/2$.*

Proof. The bound follows by noticing that it is equivalent to

$$\int_0^\alpha \int_0^x 1 - \tanh^2(u) du dx \leq \int_0^\alpha \int_0^x 1 du dx$$

which is obviously true. □

The main result of this chapter is the following theorem:

Theorem 4. *Let $y_1, y_2, \ldots, y_T$ be a sequence of symbols in $\{-1, +1\}$. Let $\boldsymbol{u}$ be a vector such that for all i, $u_i \geq 0$ and $\sum_i u_i = 1$ and suppose that over this sequence $\boldsymbol{u}$ attains loss $L = \sum_t L_t$. Then Algorithm 5 will make at most $\max\left\{\frac{2L}{\delta} + \frac{8\log T}{\delta^2}, \frac{64}{\delta^3}\right\}$ mistakes.*

Proof. The proof uses the relative entropy $D(\boldsymbol{u}||\boldsymbol{w})$ of $\boldsymbol{u}$ and $\boldsymbol{w}$ as a measure of progress. Recall that the vectors $\boldsymbol{u}$ and $\boldsymbol{w}$ have dimension $d \leq T(T-1)$. Let

$$\begin{aligned} \Phi(\boldsymbol{w}_t) &= D(\boldsymbol{u}||\boldsymbol{w}_t) = \sum_{i=1}^{d} u_i \log \frac{u_i}{w_{t,i}} \\ \Delta_t &= \Phi(\boldsymbol{w}_t) - \Phi(\boldsymbol{w}_{t+1}). \end{aligned}$$

The telescoping sum

$$\sum_{t=1}^{T} \Delta_t = \Phi(\boldsymbol{w}_1) - \Phi(\boldsymbol{w}_{T+1})$$

is not greater than $\Phi(\boldsymbol{w}_1)$ because of the non-negativity of $\Phi(\boldsymbol{w}_{T+1})$. Furthermore,

$$\Phi(\boldsymbol{w}_1) \leq \sum_{i=1}^{d} u_i \log u_i + \sum_{i=1}^{d} u_i \log(d) \leq \log d \leq 2 \log T,$$

which is true because of the non-negativity of the entropy of $\boldsymbol{u}$ and because $\boldsymbol{w}_1$ starts as a uniform distribution. Putting all these together, one obtains the upper bound

$$\sum_{t=1}^{T} \Delta_t \leq 2 \log T. \tag{3.4}$$

This upper bound on $\sum \Delta_t$ will be combined with a lower bound for the same quantity. If on round t there was no mistake, then $\Delta_t = 0$. Now assume that a mistake was made on round t. To lower bound Δ_t, we first write $\boldsymbol{w}_{t+1}$ in terms of $\boldsymbol{w}_t$:

$$w_{t+1,i} = \frac{e^{\theta_{t+1,i}}}{Z_t} = \frac{e^{\theta_{t,i} + \alpha y_t x_{t,i} + \alpha n_{t,i}}}{Z_t} = \frac{w_{t,i} e^{\alpha y_t x_{t,i} + \alpha n_{t,i}}}{Z_t}, \tag{3.5}$$

where Z_t is the normalization constant:

$$Z_t = \sum_{j=1}^{d} w_{t,j} e^{\alpha y_t x_{t,j} + \alpha n_{t,j}}. \tag{3.6}$$

Another way to define Δ_t is

$$\Delta_t = \sum_{i=1}^{d} u_i \log \frac{w_{t+1,i}}{w_{t,i}}. \tag{3.7}$$

Plugging in (3.5) in (3.7) leads to

$$\Delta_t = \sum_{i=1}^{d} u_i \log \frac{w_{t,i} e^{\alpha y_t x_{t,i} + \alpha n_{t,i}}}{Z_t w_{t,i}} = \alpha y_t \boldsymbol{u}^\top \boldsymbol{x}_t + \alpha \boldsymbol{u}^\top \boldsymbol{n}_t - \log Z_t \tag{3.8}$$

Next these three terms are bounded from below. Using Lemma 4 it is easy to bound the exponential in (3.6) by a linear function:

$$\begin{aligned} Z_t &= \sum_{i=1}^{d} w_{t,i} e^{\alpha(y_t x_{t,i} + n_{t,i})} \\ &\leq \sum_{i=1}^{d} w_{t,i} (\sinh(\alpha)(y_t x_{t,i} + n_{t,i}) + \cosh(\alpha)) \\ &= \sinh(\alpha) y_t \boldsymbol{w}_t^\top \boldsymbol{x}_t + \boldsymbol{w_t}^\top \boldsymbol{n}_t + \cosh(\alpha) \end{aligned}$$

Recall that this analysis pertains to the case when the algorithm made a mistake on round t. This means that $\sinh(\alpha) y_t \boldsymbol{w}_t^\top \boldsymbol{x}_t \leq 0$. Furthermore in Lemma 2 it was shown that either $\theta_{t,i} = 0$ or $n_{t,i} = 0$ and it was argued immediately after Lemma 1 that when $\theta_{t,i}$ is zero then $\boldsymbol{w}_t^\top \boldsymbol{v}$ does not depend on $w_{t,i}$. Therefore $\boldsymbol{w}_t^\top \boldsymbol{n}_t = 0$. These observations lead to $Z_t \leq \cosh(\alpha)$ or

$$-\log(Z_t) \geq -\log(\cosh(\alpha)). \tag{3.9}$$

Furthermore, from Hölder's inequality one gets

$$\boldsymbol{u}^\top \boldsymbol{n}_t \geq -||\boldsymbol{u}||_1 ||\boldsymbol{n}_t||_\infty = -||\boldsymbol{n}_t||_\infty \tag{3.10}$$

Moreover,

$$y_t \boldsymbol{u}^\top \boldsymbol{x}_t \geq \delta - L_t \tag{3.11}$$

by the definition of L_t, so by combining the bounds (3.9), (3.10), and (3.11), (3.8) becomes

$$\Delta_t \geq \alpha\delta - \alpha L_t - \alpha||\boldsymbol{n}_t||_\infty - \log(\cosh(\alpha)). \tag{3.12}$$

Summing up the individual lower bounds for all t and using the definitions of M_t, P_t and L leads to

$$\sum_{t=1}^{T} \Delta_t \geq M_T(\alpha\delta - \log(\cosh(\alpha))) - \alpha P_T - \alpha L.$$

Substituting the invariant (3.1) and combining this with the upper bound (3.4) leads to

$$M_T(\alpha\delta - \log\cosh(\alpha)) - \alpha {M_T}^{2/3} - \alpha L \leq 2\log T.$$

Let $z = {M_T}^{1/3}$. The inequality

$$(\alpha\delta - \log\cosh(\alpha))z^3 - \alpha z^2 - \alpha L - 2\log T \leq 0$$

is valid when z is less than the only real root of the polynomial on the left hand side. The reason this polynomial has only one real root is because

$\alpha L + 2\log T > 0$. Let ρ be this root. Then the above inequality is equivalent to $z \leq \rho$. Even though there is an exact analytical expression for ρ, much intuition can be gained from a simple upper bound [53, Theorem (30,2)]: For any $\lambda \geq 1$,

$$\rho \leq \max\left\{\left(\frac{\lambda(\alpha L + 2\log T)}{\alpha\delta - \log\cosh(\alpha)}\right)^{\frac{1}{3}}, \frac{\frac{\lambda}{\lambda-1}\alpha}{\alpha\delta - \log\cosh(\alpha)}\right\}.$$

Using this bound one gets

$$M_T \leq \max\left\{\frac{2L}{\delta} + \frac{8\log T}{\delta^2}, \frac{64}{\delta^3}\right\} \tag{3.13}$$

by setting $\alpha = \delta/2$, $\lambda = 3/2$, and using Lemma 5. □

This bound is further discussed in light of the theoretical and empirical results in the next two sections.

3.2.3 Growth Bound

It is also important to bound how quickly Algorithm 5 grows the tree. In fact, it is possible to show that the depth of the tree grows logarithmically with the number of mistakes. The proof makes use of the following lemma:

Lemma 6. *For any real $z \geq 0$,*

$$\frac{1}{\sqrt[3]{z^6 + 2z^3 + 1} - z^2} \leq \frac{3z + 2}{2}.$$

Proof. When $z \geq 0$, which is the case here, both sides are positive, so the statement of the lemma is equivalent to

$$\sqrt[3]{z^6 + 2z^3 + 1} - z^2 \geq \frac{2}{3z + 2}$$

or

$$\sqrt[3]{z^6 + 2z^3 + 1} \geq \frac{z^2(3z + 2) + 2}{3z + 2}.$$

Raising both sides to the third power and rearranging, one gets the equivalent inequality

$$(z^6 + 2z^3 + 1)(3z + 2)^3 - (3z^3 + 2z^2 + 2)^3 \geq 0$$

which when expanded yields

$$36z^5 + 48z^4 + 7z^3 + 30z^2 + 36z \geq 0.$$

The latter is obviously true because it is a polynomial with non-negative coefficients and $z \geq 0$. □

Theorem 5. *Let $A_1, A_2, \ldots$ be the sequence of trees generated by Algorithm 5 with $\beta = 2^{-1/3}$. Then for all $T \geq 2$ the depth of A_{T+1} is upper bounded by $\log_2 M_{T-1} + 4$.*

Proof. The depth of A_{T+1} is the maximum of the d_t values given by Lemma 3 over all rounds t. However, if at round t there was no mistake or $d_t \leq h_t$, then the tree is not grown. So it suffices to show that for all rounds t in which a mistake was made, the quantity

$$\left\lceil \log_\beta \left(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1} \right) - 1 \right\rceil$$

from equation (3.2) is no greater than the bound. That expression is upper-bounded by $\log_\beta(\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1})$. To get a handle on the memory requirements, it is convenient to switch to base 2 logarithms so that it is easy to relate the bound on the depth of the tree with the actual size of the tree. Using $\beta = 2^{-1/3}$, one gets that

$$d_t \leq 3 \log_2 \frac{1}{\sqrt[3]{P_{t-1}^3 + 2P_{t-1}^{3/2} + 1} - P_{t-1}}.$$

The expression inside the logarithm can be upper-bounded by $(3\sqrt{P_{t-1}} + 2)/2$ using Lemma 6 with $z = \sqrt{P_{t-1}}$. Therefore the above quantity is upper-bounded by

$$3 \log_2 \frac{3\sqrt{P_{t-1}} + 2}{2} \leq 3 \log_2 \frac{3M_{t-1}^{1/3} + 2}{2} \leq 3 \log_2 (3M_{T-1}^{1/3} + 2) - 3$$

where the first inequality comes from invariant (3.1) and the second inequality is justified because M_t is a non-decreasing sequence. Furthermore, $M_{T-1} \geq 1$, since $T \geq 2$ and the first prediction is always a zero, which counts as a mistake. Using the inequality $\log_2(3\mu + 2) \leq \log_2(5\mu)$, which is true for $\mu \geq 1$, leads to

$$3\log_2(3M_{T-1}^{1/3}+2)-3 \leq 3\log_2(5M_{T-1}^{1/3})-3 \leq \log_2 M_{T-1}+3\log_2(5)-3 < \log_2 M_{T-1}+4.$$

□

Theorem 5 says that the amount of memory needed by the tree grows only linearly with the number of mistakes. The respective growth bound given in [23] also scales linearly but with a larger constant. This is reflected in the experiments as well.

However, the comparison of the mistake bounds of the two algorithms is complex, and in general multiplicative and additive update algorithms have different merits [45]. The case here is even more intricate because $\boldsymbol{x}_t$ conceptually has twice as many dimensions as the corresponding quantity in [23] and the feature values are different. Therefore, the optimal weight vector in hindsight and its cumulative δ-hinge loss will be different for the two algorithms. The latter is likely to be larger in the formulation here because the feature values are larger. In this view, the dependence on δ^{-3} in the mistake bound presented here may not be important, since δ may be large. Notice that for the multiplicative algorithm, one can afford to push the features as close to 1 as possible without changing the mistake bound which depends only on $||\boldsymbol{x}_t||_\infty \leq 1$. However, the algorithm in [23] depends on $||\boldsymbol{x}_t||_2 \leq 1$, and their features cannot be made larger without affecting their mistake or growth bound. In Theorem 5, β is set so that the features are as large as possible while keeping the PST size linear with the number of mistakes. Finally, the mistake bound here also depends on $\log T$. Even though this dependence also exists in a lower bound [45], this lower

Table 3.1: Error Rates for PSTs on five different tasks

DATASET	ULYSSES BIT	ULYSSES A-Z	EXCEL	OUTLOOK	FIREFOX
PERCEPTRON	24.32	67.58	22.68	5.1	14.86
WINNOW	20.49	65.58	20.59	4.43	13.88

Table 3.2: Number of nodes in PSTs on five different tasks

DATASET	ULYSSES BIT	ULYSSES A-Z	EXCEL	OUTLOOK	FIREFOX
PERCEPTRON	675K	13.2M	24402	41239	21081
WINNOW	270K	10.3M	15338	25679	12662

bound assumes that an adversary selects the feature values, which is not true in this case. Nevertheless, as the next section shows, the actual number of mistakes in practice may differ markedly from these worst case bounds.

3.3 Experiments

This section reports on some experiments conducted to demonstrate the effectiveness of the proposed algorithm. Two quantities of interest are the online error rate in Table 3.1 and the size of the PST in Table 3.2. Since the data from the motivating application are not publicly available, two reproducible experiments are first described. Using the text from James Joyce's Ulysses one can define two problems: predicting the next bit and predicting the next alphabetic character in the text (non alphabetic characters were discarded in this case).

The first two columns of Tables 3.1 and 3.2 show the results of these experiments. For both tasks, the Balanced Winnow variant presented in this chapter makes fewer mistakes and grows a smaller tree than the perceptron of [23]. In all experiments, α was set to 0.1 and β was set as in theorem 5. The baseline error rates are 50% and 88% respectively.

Predicting the next letter as well as the problems from motivating application are *multiclass problems*: the sequence does not come from a binary alpha-

bet and neither Balanced Winnow nor the algorithm of [23] can directly handle alphabets with more than 2 symbols. To handle them, ideas from [17] are adapted by maintaining weights $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(k)}$ one for each class. The decision at time t is $\hat{y}_t = \arg\max_i \boldsymbol{w}^{(i)\top} \boldsymbol{x}_t$. If $\hat{y}_t \neq y_t$ then $\boldsymbol{w}^{(\hat{y}_t)}$ and $\boldsymbol{w}^{(y_t)}$ are updated by: $\theta^{(\hat{y}_t)}_{t+1,i} = \theta^{(\hat{y}_t)}_{t,i} - \alpha x_{t,i}$ and $\theta^{(y_t)}_{t+1,i} = \theta^{(y_t)}_{t,i} + \alpha x_{t,i}$ (if $|s(i)| \leq d_t$ as usual). This extension works for Balanced Winnow as well as the algorithm of [23]. A difficulty with this extension for Balanced Winnow is that it needs the normalization constant $Z^{(j)}_t$ for each class j. Strictly speaking, $Z^{(j)}_t$ cannot be computed without *a priori* knowledge of the length T of the sequence, because the classifiers conceptually have a weight for each substring of the whole sequence. However, the approximation

$$\tilde{Z}^{(j)}_t = \sum_{i \in A_t} e^{\theta^{(j)}_{t,i}}$$

works very well in practice and can be computed quickly from $\tilde{Z}^{(j)}_{t-1}$. This approximation was used in all the experiments.

In another experiment, the task was to predict the next system call that a program would execute based on the previous system calls. The data consisted of three applications: Firefox, Excel, and Outlook. For each application 40 sequences of system calls were available. The monitoring application was recording 23 different system calls. The last three columns in Tables 3.1 and 3.2 summarize the results of these experiments. For brevity, the tables report for each application the average online prediction error rate and tree size over the 40 sequences. However, Tables 3.1 and 3.2 understate the difference of the two algorithms in this problem since the multiplicative algorithm always makes fewer mistakes and grows smaller trees than the additive one for all three applications and all tested sequences. To assess statistical significance we used a paired sample two sided t-test, This showed that the differences of the reported quantities

are significant ($p < 10^{-5}$ for all tests). However, there exist sequences that can force Winnow to make more mistakes than Perceptron. In two out of the 120 sequences, Winnow initially made more mistakes and started outperforming Perceptron only after the first half of the sequence.

Finally, it is interesting to note that if instead of $P_t \leq M_t^{2/3}$ the algorithm had enforced $P_t \leq M_t^{1/2}$, one would have derived a mistake bound that always scales with δ^{-2}. By setting $\beta = 2^{-1/2}$, the growth bound would have been similar. Surprisingly, this variant has empirically no clear advantage over the additive algorithm. The analysis here allows selecting a larger β i.e. larger features which in turn allow the margin δ to be larger. This improves the mistake bound. At the same time, larger features mean that the norms of the noise vectors will be larger. Hence deciding on how much noise one should tolerate is a crucial issue. Too much noise will hurt the mistake bound and too little will cause the tree to grow very fast. A good balance can be achieved by requiring the size of the tree to scale linearly with the number of mistakes. This suggests that other choices of β in conjunction with an invariant similar to the one in (3.1) could be employed. Empirically, however, they do not work as well.

3.4 Related Work

Apart from [23], with which a comparison was made in the experiments, there are many other methods for learning PSTs, which are based on a wide range of principles such as PAC learning [57], structural risk minimization [44], and online Bayesian mixtures [65], and their generalizations [37, 55]. All these methods assume that there is an *a priori* bound on the maximum depth of the tree. For many applications, this is not known, and the algorithm should be allowed to estimate a good value of this parameter from the data.

The work of [45] contains many results and insights about multiplicative algorithms, including variants that are competitive with any vector u such that $||u||_1 \leq U$. Extending our algorithm to be competitive with any vector in this ball is also possible. Additionally, many authors, starting with [7], have noticed that Winnow's weight vector can be sparsified by zeroing the entries that have small weights compared to the largest weight in the hypothesis. This procedure is effective in practice, but comes with no theoretical guarantees. However, in the case of learning PSTs, the tree implicitly defines a partial order for the costs of setting the parameters to nonzero values. This allows one to have a sparse hypothesis and still be able to characterize the number of mistakes.

3.5 Conclusions

This chapter presented a modification of Balanced Winnow for learning PSTs in order to predict the next item in a sequence. The algorithm presented here does not rely on any assumptions about an underlying PST that generates the data. The algorithm also comes with theoretical guarantees about the number of mistakes it will make relative to the best PST determined in hindsight and about the amount of memory it will use to store its hypothesis. In all the experiments, it was found that it makes fewer mistakes and uses less memory than a Perceptron-like algorithm [23]. As the analysis shows, the damage caused by not fully growing the tree is quantified by the norm of the deterministic noise vector $\boldsymbol{n_t}$. By picking the smallest norm, one minimizes this damage. Among the norms one could use here, the smallest one is $||n_t||_\infty$, which naturally arises in the analysis of multiplicative algorithms.

CHAPTER 4

ONLINE GRADIENT DESCENT WITH IMPORTANCES

4.1 Introduction

An *importance* is a real number that describes the cost of misclassifying an example. Importances arise in many settings in machine learning such as boosting [30], covariate shift correction algorithms [39] and active learning [4, 5]. In boosting, importances are assigned to each example depending on how well that example has been classified in previous iterations. In covariate shift correction, an importance is assigned to a training example according to how close to the test distribution that example is. In the active learning framework of [4] an adaptive rejection sampling scheme is applied to each example and each retained example gets an importance equal to the inverse probability of being retained. Importances have become a de-facto language for specifying the relative importance of prediction amongst examples.

When not constrained by running time, importances can be dealt with using either black box techniques [59, 66] or direct modification of existing algorithms. Often importances can be eliminated simply by treating an example with importace h as h examples. However, as this thesis has already argued, online learning algorithms are sometimes preferred. Especially when datasets are large, algorithms such as online gradient descent are the method of choice. Here the standard approach of treating an example with importance h as h examples is typically translated into practice by computing the gradient once and then multiplying it by h. This is undesirable for large h because such an example can cause an update that is far beyond what is necessary to attain a small loss on it.

An important observation is that multiplying the gradient by h is typically *not* equivalent to doing h updates via gradient descent, because all loss functions of interest are nonlinear. The goal of this chapter is to resolve this failure by investigating alternate updates that gracefully deal with importances by taking into account the curvature of the loss function. We mainly focus on a novel set of updates that satisfy an additional *invariance* property: for all importances h, the update is equivalent to two updates with importance $h/2$. Hence we decided to call these updates *importance invariant*.

The importance invariant updates are defined via an ordinary differential equation (ODE). Even so, it was surprising to find that they have closed-form solutions for all common loss functions. Another surprising discovery was that our invariant update substantially improves the learned predictor even when $h = 1$, both in terms of the quality of best predictor after a parameter search and in terms of the robustness to parameter search, effectively reducing the desirability of searching over many different schedules of learning rates. The reason for this is that an importance invariant update smoothly interpolates between a very aggressive projection [38, 16] algorithm and a less aggressive gradient multiplier decay algorithm. All of these benefits come at near-zero computational cost.

Among the other algorithms we consider, implicit updates [45, 47] turn out to coincide with importance invariant updates for piecewise linear loss functions and provide qualitatively similar updates for other loss functions. For most other loss functions, implicit updates require a root-finding algorithm. This is quite fast in practice. The only drawback is that analysis of implicit updates for these loss functions is harder.

Finally, another reasonable way to handle importances can be derived by

adapting techniques from [24]. This approach approximates the loss by a second order Taylor expansion at the current prediction and in the direction of the update. The authors analyze updates based on this quadratic approximation for the logistic and exponential losses for which this approximation is an upper bound. These updates coincide with implicit updates for squared loss (since the quadratic approximation is exact) and are not applicable to piecewise linear losses. We will not discuss these updates any further, since they have only been analyzed for two specific loss functions.

Section 4.2 defines the problem and describes some naive but unsatisfactory approaches. Next, the importance invariant solution is proposed and a general framework for deriving invariant updates for many loss functions is presented in Section 4.3. Section 4.4 discusses some important properties of the proposed updates, such as safety. Section 4.5 briefly covers how implicit updates can handle importance weights. Section 4.6 empirically demonstrates the merits of the proposed updates even on problems without importances. Experiments where large importances naturally arises are presented in the context of active learning in Chapter 5. Section 4.7 states some conclusions.

4.2 Problem Setting and Notation

The task is to learn a good classifier from a training set of triplets (x_t, y_t, h_t), $t = 1, \dots, T$ where $\boldsymbol{x}_t \in \mathbb{R}^d$ is a vector of d features, $h_t \in \mathbb{R}_+$ is an importance, and $y_t \in \mathbb{R}$ is a label. The quality of the classifier is measured by a loss function $\ell(p, y)$ where p is the prediction of the classifier and y is the actual label. The model is linear, i.e. $p = \boldsymbol{w}^\top \boldsymbol{x}$, where $\boldsymbol{w} \in \mathbb{R}^d$ is a vector of weights. Therefore p is a real number. The domain of y will depend on the loss function and will be

mentioned when a loss function is first introduced. The goal is to find

$$\boldsymbol{w} = \operatorname*{argmin}_{w} \sum_{t=1}^{T} h_t \ell(\boldsymbol{w}^\top \boldsymbol{x}_t, y_t), \tag{4.1}$$

using a procedure similar to online gradient descent. When examples do *not* have importances the online gradient descent algorithm, shown in Algorithm 3 in page 16, is a good candidate for finding an approximate minimizer of the objective function (4.1).

When examples have importances, we maintain the semantics of the importances by adhering to the following principle: *An example with importance weight h should be treated as if it is a regular example that appears h times in the dataset.* This is a statement of both mathematical and semantic correctness. Mathematically, (4.1) states the same thing. Semantically, an example of importance h is just a convenient encoding of h identical examples. For now it is assumed that importances are integers and the learning rate sequence is constant, so all $\eta_t = \eta$. These assumptions are only for ease of exposition and are lifted in Section 4.3.

4.2.1 Some Unsatisfactory Approaches

A first approach would be to perform several passes over the data. In the i-th pass, only those examples whose importance is greater than i would be used. While this is a valid approach, it is very inefficient. Ideally each example should be presented once to the learner.

Another tempting approach is to multiply the update by the importance:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - h_t \eta \nabla_w \ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t). \tag{4.2}$$

However, this update rule does not respect the principle of Section 4.2. To see

this, consider the case $h_t = 2$. Rule 4.2 should be equivalent to

$$\begin{aligned} \boldsymbol{v} &= \boldsymbol{w}_t - \eta \nabla_w \ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t) \\ \boldsymbol{w}_{t+1} &= \boldsymbol{v} - \eta \nabla_w \ell(\boldsymbol{v}^\top \boldsymbol{x}_t, y_t) \end{aligned}$$

which is not true in general. Furthermore, the quality of this update gets worse as the importance weight gets larger. To see this, note that online gradient descent is based on approximately solving

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_w \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \eta_t \ell(\boldsymbol{w}^\top \boldsymbol{x}_t, y_t)$$

by linearizing the loss around the point $w = \boldsymbol{w}_t$:

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_w \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \eta_t \left(\ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t) + (\boldsymbol{w} - \boldsymbol{w}_t)^\top \nabla_w \ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t)\right)$$

This can be easily solved by setting the gradient of the objective to zero:

$$0 = \boldsymbol{w}_{t+1} - \boldsymbol{w}_t + \eta_t \ell(\boldsymbol{w}_t^\top \boldsymbol{x}_t, y_t)$$

and rearranging. Though this is well motivated when η_t is small, rule 4.2 implicitly assumes that the first order approximation of the loss is valid far away from its expansion point $\boldsymbol{w}_t$. Since all loss functions in machine learning are nonlinear, this assumption is simply invalid and can lead to very poor results.

Another way to see the problem is to recall that the regret bound of online gradient descent scales linearly with the square of the largest norm of the gradients computed in each round [67]. In settings like the importance weighted active learning framework of Chapter 5, multiplying the loss with the importance causes the squared norm of the gradient to be multiplied by the square of the importance. If the importances grow faster than $O(T^{1/4})$, as can happen in the algorithm of Chapter 5, then the regret bound of online gradient descent becomes $O(T)$ which is vacuous.

Another approach with good computational characteristics is *rejection sampling* according to $h/h_{\max}$. In this approach, an example with importance h is retained with probability $h/h_{\max}$, where $h_{\max}$ is an upper bound on the maximum importance in the data. Otherwise the example is discarded. The model is then learned from the unweighted set of the retained examples. However, when $h_{\max}$ is much larger than the importance of a typical example, this approach will lead to increased generalization error because too many examples will be discarded. Rejection sampling can be repaired by learning multiple predictors based upon different rejection-sampled datasets [66], but this increases computation substantially.

4.2.2 An Efficient Invariant Approach

To achieve invariance and efficiency an example with importance h can be presented h times in a row. This section will focus on the cumulative effect of this approach. This scheme respects the correctness principle and considers each example once. The only remaining question is whether the cumulative effect of h presentations in a row can be computed in time that does not scale linearly with h. While it is not obvious on how to do this for any model, the next section explains why this is possible for linear models and how to compute it.

4.3 A Framework For Working with Importances

Given a loss function of the form $\ell(p, y)$, where p is the prediction, and assuming a linear model $p = \boldsymbol{w}^\top \boldsymbol{x}$ one has that $\nabla_w \ell = \frac{\partial \ell}{\partial p} x$. Therefore all gradients of a given example point in the same direction and differ only in magnitude. Hence computing the cumulative effect of presenting the example h times in a row

amounts to computing a global scaling for x that aggregates the effects of all the gradients. Below, a simple lemma is presented that formalizes this in the case of integer importances.

Lemma 7. *Let $h \in \mathbb{N}$. Presenting example (x, y) h times in a row is equivalent to the update*

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - s(h)\boldsymbol{x}, \tag{4.3}$$

where the scaling factor $s(h)$ has this recursive form:

$$\begin{aligned} s(h+1) &= s(h) + \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x}} && (4.4) \\ s(0) &= 0. && (4.5) \end{aligned}$$

Proof. By induction on h. The base case is obvious. Now the effect of presenting the example (x, y) $h+1$ times can be computed by performing a gradient update on the vector v that results from presenting the example h times. By the induction hypothesis, this intermediate vector is

$$\boldsymbol{v} = \boldsymbol{w}_t - s(h)\boldsymbol{x}$$

and the gradient descent step is

$$\boldsymbol{w}_{t+1} = \boldsymbol{v} - \eta \nabla_w \ell(\boldsymbol{w}^\top \boldsymbol{x}, y)|_{w=v}.$$

Expanding this using the induction hypothesis we get

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - s(h)\boldsymbol{x} - \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=\boldsymbol{v}^\top \boldsymbol{x}} \boldsymbol{x} \\ &= \boldsymbol{w}_t - \left(s(h) + \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x}} \right) \boldsymbol{x}. \end{aligned}$$

□

Given a loss function, one could try to find a closed form solution to the recurrence defined by (4.4) and (4.5). For example, for squared loss $\ell(p, y) =$

$\frac{1}{2}(p-y)^2$, for which $y \in \mathbb{R}$, the recurrence is

$$s(h+1) = s(h) + \eta((\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x} - y).$$

A simple inductive argument can then verify that

$$s(h) = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{\boldsymbol{x}^\top \boldsymbol{x}}(1 - (1 - \eta \boldsymbol{x}^\top \boldsymbol{x})^h) \tag{4.6}$$

Note that when $\eta \boldsymbol{x}^\top \boldsymbol{x} < 1$, $s(h)$ asymptotically approaches the quantity that would make $\boldsymbol{w}_{t+1}^\top \boldsymbol{x} = y$. This behavior is more desirable than that of multiplying the gradient by the importance weight.

Update 4.6 is a first breakthrough that allows to compute $s(h)$ fast. However, it is restricted to integer importances. Moreover, other loss functions do not yield a recurrence with a closed form solution. To overcome these problems, another ingredient is necessary. Starting from (4.6), is it instructive to think about the consequences of presenting an example many times. To compensate, the learning rate will also need to be adjusted so that its effect is split among the updates. Suppose that the algorithm sees an example a factor of n times more using a learning rate that is smaller by a factor of n. This can be simulated in constant time using (4.6) with hn and η/n in place of h and η, respectively. Letting n grow large, the quantity of interest is

$$\lim_{n \to \infty} \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{\boldsymbol{x}^\top \boldsymbol{x}} \left(1 - \left(1 - \frac{\eta \boldsymbol{x}^\top \boldsymbol{x}}{n}\right)^{nh}\right).$$

Using that, $\lim_{n \to \infty}(1 + z/n)^n = e^z$ leads to

$$s(h) = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{\boldsymbol{x}^\top \boldsymbol{x}} \left(1 - \exp(-h\eta \boldsymbol{x}^\top \boldsymbol{x})\right). \tag{4.7}$$

In the above derivation, the gradient descent process happens in continuous time, which is the limit of the discrete process as the learning rate becomes infinitesimal. This idea is now generalized to derive updates for other loss functions.

Theorem 6. *The limit of the gradient descent process as the learning rate becomes infinitesimal for an example with importance $h \in \mathbb{R}_+$ is equal to the update*

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - s(h)\boldsymbol{x},$$

where the scaling factor $s(h)$ satisfies the differential equation:

$$s'(h) = \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x}}, \quad s(0) = 0. \tag{4.8}$$

Proof. In accordance with the proof of Lemma 7, it is possible to compute the effect of an importance $h + \epsilon$ assuming knowledge of the effect of an importance h by performing an additional gradient step with the learning rate appropriately scaled by ϵ:

$$s(h + \epsilon) = s(h) + \epsilon\eta \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x}}.$$

Rearranging leads to

$$\frac{s(h + \epsilon) - s(h)}{\epsilon} = \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x}}.$$

Taking the limit as ϵ approaches 0 gives the result. The initial condition makes sure that a zero importance has no effect in the update. □

This theorem will be the framework for deriving updates for many loss functions. Plugging a loss function in (4.8) gives an ODE whose solution is the result of a continuous gradient descent process. The ODE can be easily solved by separation of variables.

As a sanity check for squared loss we have $\frac{\partial \ell}{\partial p} = p - y$ and (4.8) gives

$$s'(h) = \eta((\boldsymbol{w}_t - s(h)\boldsymbol{x})^\top \boldsymbol{x} - y), \quad s(0) = 0,$$

a linear ODE, whose solution rederives (4.7).

Table 4.1: Importance Invariant Updates for Various Loss Functions

Loss	$\ell(p, y)$	Invariant Update $s(h)$
Squared	$\frac{1}{2}(y-p)^2$	$\frac{p-y}{\|\|x\|\|^2}\left(1-e^{-h\eta\|\|x\|\|^2}\right)$
Logistic	$\log(1+e^{-yp})$	$\frac{W(e^{h\eta\|\|x\|\|^2+yp+e^{yp}})-h\eta\|\|x\|\|^2-e^{yp}}{y\|\|x\|\|^2}$
Exponential	e^{-yp}	$\frac{py-\log(h\eta\|\|x\|\|^2+e^{py})}{\|\|x\|\|^2 y}$
Logarithmic	$y\log\frac{y}{p}+(1-y)\log\frac{1-y}{1-p}$	if $y=0$ $\frac{p-1+\sqrt{(p-1)^2+2h\eta\|\|x\|\|^2}}{\|\|x\|\|^2}$ if $y=1$ $\frac{p-\sqrt{p^2+2h\eta\|\|x\|\|^2}}{\|\|x\|\|^2}$
Hellinger	$1-\sqrt{py}-\sqrt{(1-p)(1-y)}$	if $y=0$ $\frac{p-1+\frac{1}{4}(12h\eta\|\|x\|\|^2+8(1-p)^{3/2})^{2/3}}{\|\|x\|\|^2}$ if $y=1$ $\frac{p-\frac{1}{4}(12h\eta\|\|x\|\|^2+8p^{3/2})^{2/3}}{\|\|x\|\|^2}$
Hinge	$\max(0, 1-yp)$	$\min\left(-yh\eta, \frac{p-y}{\|\|x\|\|^2}\right)$
τ-Quantile	if $y>p$ $\tau(y-p)$ if $y\le p$ $(1-\tau)(p-y)$	if $y>p$ $\min(-\tau h\eta, \frac{p-y}{\|\|x\|\|^2})$ if $y\le p$ $\min((1-\tau)h\eta, \frac{p-y}{\|\|x\|\|^2})$

4.3.1 Other Loss Functions

Using (4.8) as a framework, one can derive step sizes for many popular loss functions as summarized in Table 4.1.

For the logistic loss, for which $y \in \{-1, +1\}$, the solution involves the Lambert W function $W(z)e^{W(z)} = z$. The solution can be verified using $W'(z) = \frac{W(z)}{z(1+W(z))}$. The exponential loss, for which $y \in \{-1, +1\}$, also fits nicely into this framework.

For the logarithmic loss, for which $y \in [0, 1]$, the ODE has no explicit form for all $y \in [0, 1]$. The table presents the common case $y \in \{0, 1\}$. In this case each value of y gives rise to an ODE whose solution has an explicit form. Note that here the ODE solutions satisfy a second-degree equation, hence each branch has two solutions. The selected branch is the one satisfying $s'(0) = \eta\frac{\partial \ell}{\partial p}$. To avoid an infinite loss when using the logarithmic loss directly with a linear model, one should clip the predictions in an interval $[p_{\min}, p_{\max}]$ so that $p_{\min} > 0$ and $p_{\max} < 1$. In this case updates should use $\min(h, h')$ as the importance, where

h' is the importance that leads to an update that reaches the clipping point. Alternatively, one could use a *link function* such as $\sigma(p) = (1 + \tanh(p))/2$, where $p = \boldsymbol{w}_t^\top \boldsymbol{x}_t$. As in the case without the link function, we get an explicit form for $y \in \{0, 1\}$, which is

$$s(h) = \begin{cases} \frac{p + \frac{1}{2}\log(\omega_0)}{||\boldsymbol{x}||^2} & \text{if } y = 0 \\ \frac{p - \frac{1}{2}\log(\omega_1)}{||\boldsymbol{x}||^2} & \text{if } y = 1, \end{cases}$$

where

$$\begin{aligned} \omega_0 &= W(\exp(e^{2p} + 2p + 4h\eta||\boldsymbol{x}||^2)) \\ \omega_1 &= W(\exp(e^{-2p} - 2p + 4h\eta||\boldsymbol{x}||^2)). \end{aligned}$$

As in the case of logistic loss, $W(z)$ is the Lambert W function.

A similar situation arises for the Hellinger loss, for which $y \in [0, 1]$. The solution to (4.8) has no simple form for all $y \in [0, 1]$ but for $y \in \{0, 1\}$ we get the expressions in Table 4.1.

Hinge Loss and Quantile Loss

Two other commonly used loss functions are the hinge loss and the τ-quantile loss, where $\tau \in [0, 1]$ is a parameter. These are differentiable everywhere except at one point where the subdifferential contains zero.

For the hinge loss, a valid expression for (4.8) is

$$s'(h) = \begin{cases} -\eta y & y(\boldsymbol{w} - s(h)\boldsymbol{x})^\top \boldsymbol{x} < 1 \\ 0 & y(\boldsymbol{w} - s(h)\boldsymbol{x})^\top \boldsymbol{x} \geq 1. \end{cases}$$

The first branch (together with $s(0) = 0$) gives $s(h) = -yh\eta$ for $y(\boldsymbol{w} + yh\eta\boldsymbol{x})^\top \boldsymbol{x} < 1$. Otherwise, i.e. when

$$h \geq h_{\text{hinge}} = \frac{1 - y\boldsymbol{w}^\top \boldsymbol{x}}{\eta \boldsymbol{x}^\top \boldsymbol{x}},$$

$s(h)$ is a constant. Here h_{hinge} is the importance that would make the updated prediction lie at the hinge. To maintain continuity at h_{hinge} we set $s(h) = -yh_{\text{hinge}}\eta$. In conclusion,

$$s(h) = \min(-yh\eta, -yh_{\text{hinge}}\eta).$$

This matches intuition when one considers the limit of infinitely many infinitely small updates: For large importance weights, the process will bring the prediction up to y and make no further progress.

The quantile loss is similar. The update rule first computes the importance weight h' that would take the updated prediction at the point of nondifferentiability, then multiplies the gradient by $\min(h, h')$.

4.3.2 Variable Learning Rate

To handle a decaying learning rate η_t, one just needs to modify (4.8) slightly. Let $\eta_t(u)$ be the value of the learning rate u timesteps after time t. Then (4.8) becomes

$$s'(h) = \eta_t(h) \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(h)\boldsymbol{x}_t)^\top \boldsymbol{x}_t}, \quad s(0) = 0.$$

The solutions in this case are not qualitatively any different from the solutions of (4.8). After the separation of variables s and h, integration on the side containing dh will have to take into account $\eta_t(h)$. Hence the modification boils down to replacing the occurrences of $h\eta$ with $\int_0^h \eta_t(u)du$. Again, for popular choices of learning rate such as $\eta_t(u) = (t+u)^{-q}$ with $q = \frac{1}{2}$ or $q = 1$, this has a closed form.

4.3.3 Regularization

Theorem 6 can be modified to handle losses of the form $\ell(\boldsymbol{w}^\top \boldsymbol{x}, y) + \frac{\lambda}{2}||w||^2$. However, the resulting differential equation is considerably harder, and only

the case of squared loss has an easy closed form solution. An alternative way of incorporating regularization is based on splitting [27]. First perform h unconstrained steps using the closed form solution, then compute the effect of regularization:

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_{w} \frac{1}{2}||\boldsymbol{w} - (\boldsymbol{w}_t - s(h)\boldsymbol{x})||^2 + \frac{h\eta\lambda}{2}||\boldsymbol{w}||^2.$$

Note that we apply all h regularizers at once. The solution to the above optimization problem is

$$\boldsymbol{w}_{t+1} = \frac{\boldsymbol{w}_t - s(h)\boldsymbol{x}_t}{1 + h\eta\lambda}.$$

This approach can also handle other regularizers such as $\lambda||\boldsymbol{w}||_1$, leading to a truncated gradient update [49].

4.4 Properties of the Updates

4.4.1 Invariance

We first show that the updates satisfy an invariance property. Updating with importance $a + b$ is is equivalent to an update with importance a immediately followed by an update with importance b. It is convenient to explicitly state the dependence on the prediction p, by writing $s(p, h)$ instead of $s(h)$. The following theorem states this.

Theorem 7. *Let $s(p, h)$ be the solution of*

$$\frac{\partial s}{\partial h} = \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=\boldsymbol{w}^\top\boldsymbol{x} - s(p,h)\boldsymbol{x}^\top\boldsymbol{x}}, \quad s(p, 0) = 0,$$

where ℓ is a continuously differentiable loss. Then

$$s(p, a + b) = s(p, a) + s(p - s(p, a)\boldsymbol{x}^\top\boldsymbol{x}, b).$$

Proof. For convenience let $f(z) = \eta \left.\frac{\partial \ell}{\partial p}\right|_{p=z}$ and $n = \boldsymbol{x}^\top \boldsymbol{x}$. The theorem says that if f is continuous and s satisfies

$$\frac{\partial s}{\partial h} = f(p - ns(p,h))$$
$$s(p,0) = 0,$$

then

$$s(p, a+b) = s(p,a) + s(p - ns(p,a), b). \tag{4.9}$$

Now fix an arbitrary value of a, and consider the pair of single-variable functions

$$u(b) = s(p, a+b)$$
$$v(b) = s(p,a) + s(p - ns(p,a), b).$$

These satisfy

$$u(0) = v(0) = s(p,a)$$

$$\frac{du}{db} = \left.\frac{\partial s}{\partial h}\right|_{(p,a+b)} = f(p - ns(p, a+b)) = f(p - nu(b))$$
$$\frac{dv}{db} = \left.\frac{\partial s}{\partial h}\right|_{(p-ns(p,a),b)} = f(p - ns(p,a) - ns(p - ns(p,a), b)) = f(p - nv(b)).$$

In other words, both u and v are solutions of the ordinary differential equation

$$\frac{dw}{db} = f(p - nw(b))$$

with initial condition $w(0) = s(p,a)$. Since f satisfies the hypotheses of the existence and uniqueness theorem for ordinary differential equations, it is valid to conclude that $u(b) = v(b)$, which verifies (4.9). □

The existence and uniqueness theorem for ODEs, used in the proof, shows that $s(p,h)$ is the unique function with the invariance propery, but does not shed any light on further properties of $s(p,h)$.

4.4.2 Safety

For some loss functions such as squared loss, hinge loss, and quantile loss, the residual $\boldsymbol{w}_t^\top \boldsymbol{x}_t - y_t$ is an indicator of whether the learner is overestimating or underestimating the target. An update is said to be *safe* if

$$\frac{\boldsymbol{w}_{t+1}^\top \boldsymbol{x}_t - y_t}{\boldsymbol{w}_t^\top \boldsymbol{x}_t - y_t} \geq 0$$

whenever $(\boldsymbol{w}_t^\top \boldsymbol{x}_t - y_t) \neq 0$. Since the residual does not change sign after a safe update, the weights cannot wildly oscillate even when the learning rate is very aggressive.

Standard gradient descent is not safe, and importance-invariant step sizes always are. We can verify this for each loss function separately

For the squared loss:

$$\frac{\boldsymbol{w}_{t+1}^\top \boldsymbol{x} - y}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} + \frac{y - \boldsymbol{w}_t^\top \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} \left(1 - e^{-h\eta \boldsymbol{x}^\top \boldsymbol{x}}\right) \boldsymbol{x}^\top \boldsymbol{x} - y}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} = e^{-h\eta \boldsymbol{x}^\top \boldsymbol{x}} > 0.$$

For the hinge loss there are two cases. The first case is when $-yh\eta \geq \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{||\boldsymbol{x}||^2}$. In this case the update is

$$\frac{\boldsymbol{w}_{t+1}^\top \boldsymbol{x} - y}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{||\boldsymbol{x}||^2} ||\boldsymbol{x}||^2 - y}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} = 0.$$

The other case can be written as $yh\eta||\boldsymbol{x}||^2 > y - \boldsymbol{w}_t^\top \boldsymbol{x}$ which leads to

$$\frac{\boldsymbol{w}_{t+1}^\top \boldsymbol{x} - y}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} + yh\eta||\boldsymbol{x}||^2 - y}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} > \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y + y - \boldsymbol{w}_t^\top \boldsymbol{x}}{\boldsymbol{w}_t^\top \boldsymbol{x} - y} = 0.$$

Hence the update is safe in both cases.

The quantile loss is similar to the hinge loss. Again there are two main cases since we need to distinguish between $\boldsymbol{w}_t^\top \boldsymbol{x} > y$ and $\boldsymbol{w}_t^\top \boldsymbol{x} < y$. However, each case has two sub-cases: in the first case we need to check whether $(1 - \tau)h\eta$ is larger or smaller than the step size that would make the updated prediction equal to y. In the second case the sub-cases are similar but the relevant quantity is $-\tau h\eta$.

4.4.3 Fallback Regret Analysis

Ideally, one would like to be able to show a result similar to that of Section 2.3.3. For example, the regret of online gradient descent using the proposed updates grows sublinearly with the number of examples. However, this certainly cannot work when the sequence of importances is arbitrary. For example in a sequence whose first importance is arbitrarily large, it is easy to see that no statement can be made about the regret of the algorithm.

This section provides a fallback analysis for the case $h_t = 1$. For simplicity the results are only shown for squared loss and $||\boldsymbol{x}_t|| = 1$ for all t. However, this can be extended to other losses, as a Taylor expansion of each update around $\eta = 0$ shows that to first order, it is equivalent to online gradient descent. Hence one should expect a regret analysis similar to the one achieved by the underlying learning rate schedule. The proof of the theorem uses the following lemma multiple times.

Lemma 8. *For all $y > 0$,*

$$\frac{1}{y+1} \leq 1 - \exp\left(-\frac{1}{y}\right) \leq \frac{1}{y}.$$

Proof. Consider the linear approximation of the function $\exp(z)$ at the point $z = 0$. By Taylor's theorem this is $\exp(0) + (z - 0)\exp(0) = 1 + z$. Since $\exp(z)$ is convex,

$$\exp(z) \geq 1 + z. \tag{4.10}$$

To prove the second inequality of the lemma, we simply substitute $z = -1/y$ in (4.10) and rearrange. To prove the first inequality, we plug in $z = 1/y$ in (4.10) and have

$$\exp\left(\frac{1}{y}\right) \geq 1 + \frac{1}{y}$$

Since $y > 0$ both sides are positive, so this is equivalent to

$$\exp\left(-\frac{1}{y}\right) \leq \frac{y}{y+1} \tag{4.11}$$

Adding 1 to both sides and rearranging leads to

$$1 - \frac{y}{y+1} \leq 1 - \exp\left(-\frac{1}{y}\right)$$

which proves the first inequality and the lemma. □

Theorem 8. *If $\ell(p, y) = (p-y)^2$ and $||\boldsymbol{x}_t|| = 1$ for all t, then the importance invariant update attains a regret of $O(\sqrt{T})$ when $\eta_t = t^{-1/2}$ and a regret of $O(\log(T))$ when $\eta_t = t^{-1}$.*

Proof. First, consider the case when $\eta_t = t^{-1/2}$ and therefore the step sizes are $\eta'_t = 1 - \exp(-t^{-1/2})$. In general the regret of online gradient descent depends on the step sizes via [67]:

$$\frac{1}{\eta'_T} + \sum_{t=1}^{T} \eta'_t = \frac{1}{1 - \exp(-T^{-1/2})} + \sum_{t=1}^{T} \left(1 - \exp\left(-\frac{1}{\sqrt{t}}\right)\right).$$

Using Lemma 8 the first term is bounded as

$$\frac{1}{1 - \exp(-T^{-1/2})} \leq \sqrt{T+1} \leq \sqrt{T} + 1.$$

Using Lemma 8 again, the summation is bounded by

$$\sum_{t=1}^{T} \left(1 - \exp\left(-\frac{1}{\sqrt{t}}\right)\right) \leq \sum_{t=1}^{T} \frac{1}{\sqrt{t}} \leq 1 + \int_1^T \frac{1}{\sqrt{t}} dt \leq 2\sqrt{T} - 1.$$

Putting the two bounds together:

$$\frac{1}{\eta'_T} + \sum_{t=1}^{T} \eta'_t \leq 3\sqrt{T}.$$

Now, consider the case $\eta_t = t^{-1}$. This type of schedule is known [62] to give logarithmic regret for strongly convex functions such as squared loss. The same

holds true for the update rule presented here. In general, for a 1-strongly convex loss such as squared loss, regret depends on [62]:

$$\sum_{t=1}^{T}\left(\frac{1}{\eta'_t}-\frac{1}{\eta'_{t-1}}-1\right)+\sum_{t=1}^{T}\eta'_t=$$

$$\sum_{t=1}^{T}\left(\frac{1}{1-\exp(-t^{-1})}-\frac{1}{1-\exp(-(t-1)^{-1})}-1\right)+\sum_{t=1}^{T}\left(1-\exp(-t^{-1})\right)$$

Using Lemma 8, the second sum can be bounded as

$$\sum_{t=1}^{T}\left(1-\exp(-t^{-1})\right)\leq\sum_{t=1}^{T}\frac{1}{t}\leq 1+\int_{t=1}^{T}\frac{1}{t}dt=\log(T)+1. \tag{4.12}$$

The first sum is telescoping leaving

$$\sum_{t=1}^{T}\left(\frac{1}{1-\exp(-t^{-1})}-\frac{1}{1-\exp(-(t-1)^{-1})}-1\right)=\frac{1}{1-\exp(-T^{-1})}-1-T.$$

A final application of Lemma 8 leads to

$$\sum_{t=1}^{T}\left(\frac{1}{1-\exp(-t^{-1})}-\frac{1}{1-\exp(-(t-1)^{-1})}-1\right)\leq 0$$

Combining this with (4.12) we obtain that

$$\sum_{t=1}^{T}\left(\frac{1}{\eta'_t}-\frac{1}{\eta'_{t-1}}-1\right)+\sum_{t=1}^{T}\eta'_t\leq\log(T)+1,$$

which is the regret obtained by the original $\eta_t=t^{-1}$ update. □

4.5 Implicit Importance-Weighted Updates

Implicit updates, first proposed in [45] and recently analyzed in [47], provide an alternative way for handling importances. An implicit update sets:

$$\boldsymbol{w}_{t+1}=\operatorname{argmin}\frac{1}{2}||\boldsymbol{w}-\boldsymbol{w}_t||^2+\lambda\ell(\boldsymbol{w}^\top\boldsymbol{x},y),$$

where λ is a free parameter similar to the learning rate in interpretation. Finding the minimizing $\boldsymbol{w}$ generally requires an iterative root-finding algorithm. This is

perhaps an order of magnitude more expensive than the closed form updates derived above, although often overshadowed by the update itself.

To adapt implicit updates for large importances one can simply use $\lambda_t = \eta h_t$, or $\lambda_t = \int_0^{h_t} \eta_t(u)du$ as in section 4.3.2, yielding the "Imp^2" algorithm (for implicit importance). Imp^2 has qualitatively similar properties, satisfying safety, as defined in Section 4.4.2. When all importances are 1, Imp^2 satisfies the same regret bounds as online gradient descent [47]. However it does not satisfy the invariance property of Section 4.4.1 nor has a closed-form update. In fact, Imp^2 has a closed form solution only for squared loss, hinge loss and quantile loss. For the latter two loss functions, it is equivalent to the importance-invariant update. The next subsections show the derivations for these cases. The results on squared loss and hinge loss are known [47, 16], but the result for quantile loss is new.

4.5.1 Squared Loss

For the squared loss we have

$$\boldsymbol{w}_{t+1} = \arg\min \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \frac{\lambda}{2}(\boldsymbol{w}^\top \boldsymbol{x} - y)^2.$$

Taking the derivative of the objective with respect to w and setting to zero we get

$$\boldsymbol{w} = \boldsymbol{w}_t - \lambda(\boldsymbol{w}^\top \boldsymbol{x} - y)x. \tag{4.13}$$

One can solve for $\boldsymbol{w}^\top \boldsymbol{x}$ by taking the inner product of both sides with x:

$$\boldsymbol{w}^\top \boldsymbol{x} = \boldsymbol{w}_t^\top \boldsymbol{x} - \lambda(\boldsymbol{w}^\top \boldsymbol{x} - y)\boldsymbol{x}^\top \boldsymbol{x}$$

$$\boldsymbol{w}^\top \boldsymbol{x} = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} + \lambda y \boldsymbol{x}^\top \boldsymbol{x}}{1 + \lambda \boldsymbol{x}^\top \boldsymbol{x}}.$$

Now substituting back in (4.13) yields

$$\boldsymbol{w} = \boldsymbol{w}_t - \frac{\lambda(\boldsymbol{w}_t^\top \boldsymbol{x} - y)}{1 + \lambda \boldsymbol{x}^\top \boldsymbol{x}} \boldsymbol{x}.$$

4.5.2 Hinge Loss

The implicit update for the hinge loss comes from the solution of

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \arg\min \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \lambda\xi \\ \text{subject to} \quad & \xi \geq 0 \\ & \xi \geq 1 - y\boldsymbol{w}^\top \boldsymbol{x} \end{aligned}$$

This update has been considered by [16] under the name PA-I. They show that the solution to the above optimization problem is

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y \min(\lambda, \frac{1 - y\boldsymbol{w}_t \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}}) \boldsymbol{x},$$

which is the same as our importance-invariant update. However, in their algorithm λ remains fixed for all t, while here $\lambda = h_t \eta_t$ varies as t varies.

4.5.3 Quantile Loss

For the quantile loss, the update can be written as:

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \arg\min \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \lambda\xi \\ \text{subject to} \quad & \xi \geq \tau(y - \boldsymbol{w}^\top x) \\ & \xi \geq (1 - \tau)(\boldsymbol{w}^\top x - y) \end{aligned}$$

To find $\boldsymbol{w}_{t+1}$ we introduce the Lagrangian:

$$\begin{aligned} L(w, \xi, \mu) &= \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \lambda\xi + \\ & + \mu_1(\tau(y - \boldsymbol{w}^\top \boldsymbol{x}) - \xi) + \\ & + \mu_2((1 - \tau)(\boldsymbol{w}^\top \boldsymbol{x} - y) - \xi), \end{aligned}$$

with Lagrange multipliers $\mu_1 \geq 0$, $\mu_2 \geq 0$. Setting the derivative of L to zero with respect to $\boldsymbol{w}$ and ξ leads to

$$\boldsymbol{w} = \boldsymbol{w}_t + \mu_1 \tau \boldsymbol{x} - \mu_2 (1-\tau)\boldsymbol{x} \tag{4.14}$$

$$\mu_1 + \mu_2 = \lambda, \tag{4.15}$$

The following cases are now distinguished:

1. Case $\boldsymbol{w}^\top \boldsymbol{x} > y$. This means that $(1-\tau)(\boldsymbol{w}^\top \boldsymbol{x} - y) > 0$, therefore the second constraint suggests $\xi > 0$. This implies that $\tau(y - \boldsymbol{w}^\top \boldsymbol{x}) - \xi < 0$, thus the first constraint is not tight. Recall that the minimizer of any constrained optimization problem must satisfy the Karush-Kuhn-Tucker complementary slackness conditions. Using these conditions here we get $\mu_1(\tau(y - \boldsymbol{w}^\top \boldsymbol{x}) - \xi) = 0$ and conclude that in this case $\mu_1 = 0$ and hence $\mu_2 = \lambda$. Therefore (4.14) becomes $\boldsymbol{w} = \boldsymbol{w}_t - \lambda(1-\tau)\boldsymbol{x}$. Using this, it is possible to write the condition $\boldsymbol{w}^\top \boldsymbol{x} > y$ in terms of the *a priori* known $\boldsymbol{w}_t$ as:
$$\frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{(1-\tau)\boldsymbol{x}^\top \boldsymbol{x}} > \lambda.$$

2. Case $\boldsymbol{w}^\top \boldsymbol{x} < y$. In a manner similar to the previous case, one can show that $\mu_2 = 0$ using the complementary slackness condition $\mu_2((1-\tau)(\boldsymbol{w}^\top \boldsymbol{x} - y) - \xi) = 0$, therefore $\mu_1 = \lambda$ and $\boldsymbol{w} = \boldsymbol{w}_t + \lambda\tau\boldsymbol{x}$. Again the condition ($\boldsymbol{w}^\top \boldsymbol{x} > y$) can be written in terms of $\boldsymbol{w}_t$ as:
$$\frac{y - \boldsymbol{w}_t^\top \boldsymbol{x}}{\tau \boldsymbol{x}^\top \boldsymbol{x}} > \lambda.$$

3. Case $\boldsymbol{w}^\top \boldsymbol{x} = y$. Plugging in $\boldsymbol{w}$ from (4.14) and rearranging the terms, we have
$$\boldsymbol{w}_t^\top \boldsymbol{x} + (\tau(\mu_1 + \mu_2) - \mu_2)\boldsymbol{x}^\top \boldsymbol{x} = y.$$

Now one uses (4.15) to get an equation involving only μ_2, which yields

$$\mu_2 = \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{\boldsymbol{x}^\top \boldsymbol{x}} + \tau\lambda,$$

hence

$$\mu_1 = \lambda - \mu_2 = \frac{y - \boldsymbol{w}_t^\top \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} + (1-\tau)\lambda.$$

Plugging these back to the condition on w from the Lagrangian leads to

$$w = \boldsymbol{w}_t - \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{\boldsymbol{x}^\top \boldsymbol{x}} \boldsymbol{x}.$$

To sum up, the implicit update for the quantile loss is

$$w = \begin{cases} \boldsymbol{w}_t - \lambda(1-\tau)\boldsymbol{x}, & \frac{\boldsymbol{w}_t^\top x - y}{(1-\tau)x^\top x} > \lambda \\ \boldsymbol{w}_t + \lambda\tau\boldsymbol{x}, & \frac{y - \boldsymbol{w}_t^\top x}{\tau x^\top x} > \lambda \\ \boldsymbol{w}_t - \frac{\boldsymbol{w}_t^\top \boldsymbol{x} - y}{\boldsymbol{x}^\top \boldsymbol{x}} \boldsymbol{x} & \text{otherwise,} \end{cases}$$

which is the same as the importance invariant update.

4.6 Experiments

This section presents empirical results on four text classification datasets. The dataset `rcv1` is a modified version [48] of RCV1 [51], `astro` is from [41], `spam` was created from the TREC 2005 spam public corpora, `webspam` is from the PASCAL large scale learning challenge. In all experiments, a single pass through the training set was performed and the error on the test set was reported. Many runs were performed each with its own learning rate schedule of the form

$$\eta_t = \frac{\mu}{||\boldsymbol{x}_t||^2} \left(\frac{t_0}{t + t_0} \right)^q$$

with $(\mu, t_0, q) \in \{2^i\}_{i=0}^{10} \times \{10^i\}_{i=0}^{8} \times \{0.5, 1\}$.

The experiments treated all examples as having an importance weight of 1. Experiments with large importances are conducted in Section 5.5 in the context of the active learning algorithm of Chapter 5. We compared standard online gradient vs. invariant and implicit updates on four loss functions: squared, logistic, hinge and quantile ($\tau = 0.5$) loss. The purpose of these experiments was to highlight the robustness of the invariant updates: We found that they yield good generalization with little search for a good learning rate schedule (also noted in [47] for implicit updates). Before discussing this further, Table 4.2 shows the test accuracy of the hypotheses learned by each update after exhaustively searching over the learning rate schedule. For `astro` and `rcv1` the differences are very small. The documents in the `spam` dataset are not processed with TF-IDF, which might explain the larger improvement with invariant and implicit updates. The results on the `webspam` dataset were initially puzzling, but it has been verified that this is not a failure to optimize well; on the contrary, the proposed updates attain smaller *progressive validation loss* [8] than standard online gradient descent on the training data. Since progessive validation loss behaves like a test set loss [8], this is evidence that the `webspam` test set has a different distribution from the training set.[1]

To illustrate robustness, results are presented in two ways. First, in Figures 4.1 and 4.2, four scatter plots are shown where each point is a learning rate schedule and its coordinates are the accuracy of the learned hypothesis with and without the invariant updates. The scatter plots include various loss functions and datasets. Other combinations of loss function and dataset look very similar and are omitted.

The plots only show the cases in which both learning rates achieve accuracy

[1]The test set consisted of the last 50000 examples of the original training set. The real test labels are not public.

Table 4.2: Test accuracies (grid search over schedules)

Dataset	Loss	Invariant	Imp^2	Standard
astro	hinge	0.96626	Same	0.96694
	quantile	0.96629	Same	0.96703
	logistic	0.96494	0.96485	0.96432
	squared	0.96463	0.96429	0.96469
rcv1	hinge	0.94872	Same	0.94838
	quantile	0.94846	Same	0.94859
	logistic	0.94704	0.94682	0.94743
	squared	0.94769	0.94799	0.94790
spam	hinge	0.97626	Same	0.97411
	quantile	0.97524	Same	0.97484
	logistic	0.96676	0.97982	0.97603
	squared	0.97609	0.97614	0.97563
webspam	hinge	0.98936	Same	0.99142
	quantile	0.98908	Same	0.99088
	logistic	0.99094	0.99038	0.9923
	squared	0.98960	0.98966	0.99218

Table 4.3: Fraction of schedules with near optimal error

Loss	Invariant	Standard
hinge	0.337	0.039
logistic	0.109	0.050
quantile	0.361	0.053
squared	0.306	0.031

above 0.9 and there are virtually no schedules for which the difference in accuracy is larger than 0.1. Among these cases, the vast majority of experiments are clustered under the $y = x$ line and towards the extreme values of the x-axis. Consequently, when using the invariant update, many schedules provide excellent performance.

To make this clearer, a second way of viewing this result is provided in Table 4.3. This table reports the fraction of learning rate schedules that achieve generalization accuracy to within 0.001 of the best learning rate schedule on average across all four datasets. For loss functions for which there is a notion of

overshooting and which can benefit from a safe update, an order of magnitude improvement is observed in the number of schedules that converge to near optimal performance.

4.7 Conclusions

This chapter described how to tune online gradient descent learning algorithms for various losses so they efficiently incorporate importance information. Such information is needed for applications in boosting, active learning, transfer learning, and learning reductions. The essential lesson here is that taking into account the properties of the loss function can be done cheaply and can provide great benefits even when examples have the same importance.

Motivated by an invariance property, new classes of updates were proposed that improve the standard update rule even when all examples have equal importance, yielding better prediction performance while simultaneously reducing the value of searching for good parameters for the learning rate schedule. Further experiments that we conducted but did not report here showed that invariant updates even improve the performance of adaptive gradient descent methods such as those proposed in [10, 26]. Since invariant updates are computationally free, they are easy to incorporate in existing software.

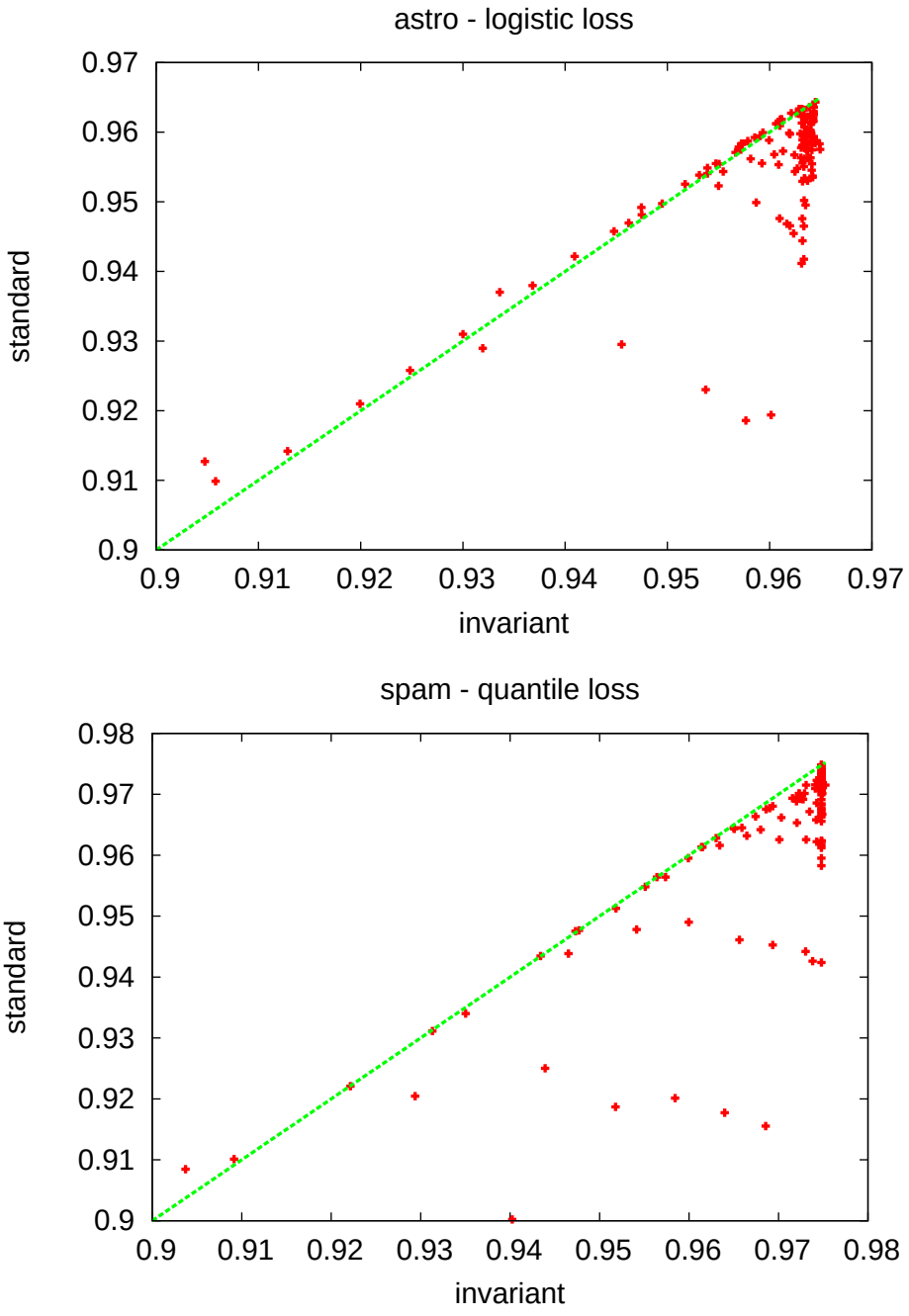

Figure 4.1: Scatter plots showing test accuracy with two different updates for various datasets and losses

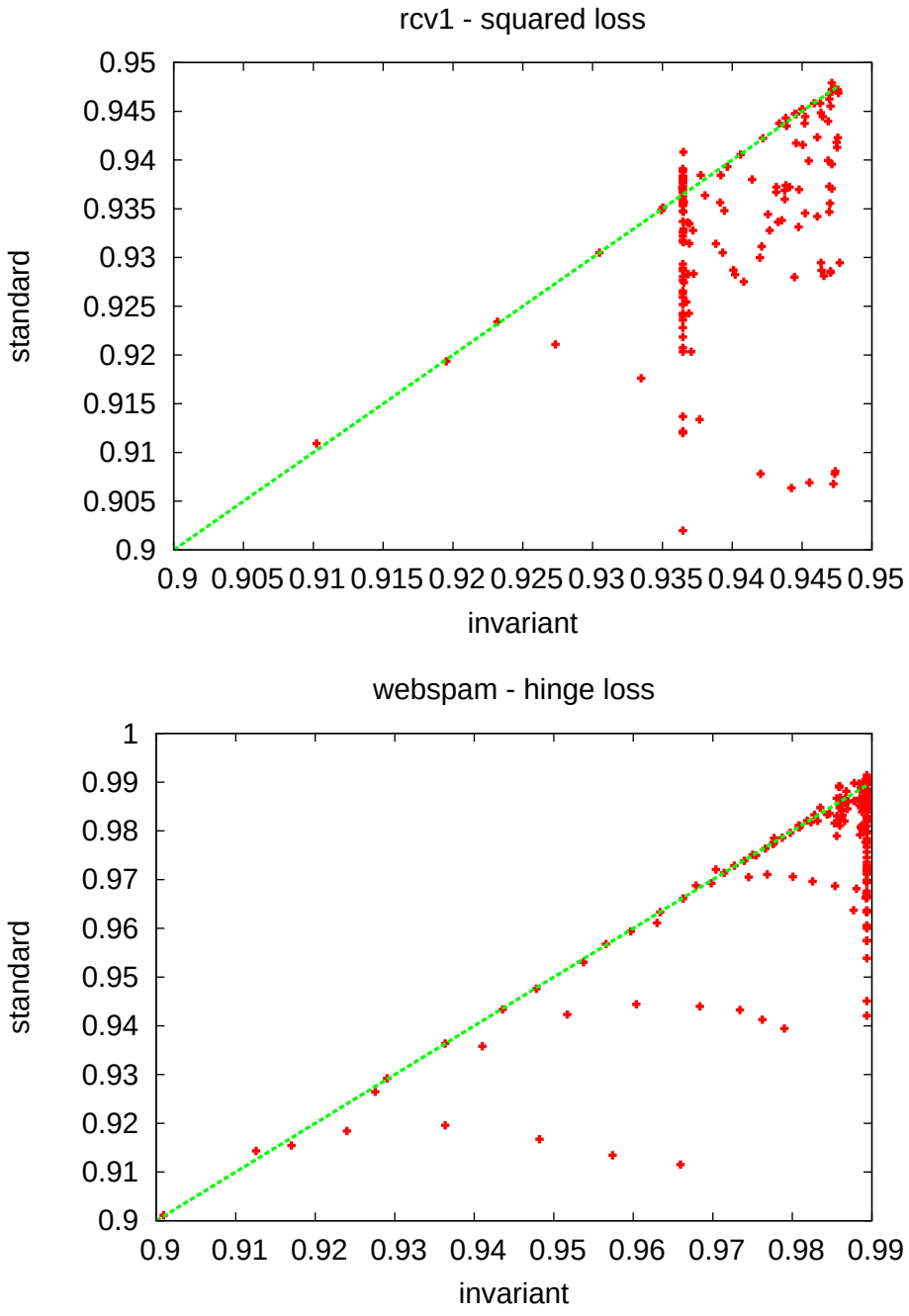

Figure 4.2: Scatter plots showing test accuracy with two different updates for various datasets and losses

CHAPTER 5

FAST AGNOSTIC ACTIVE LEARNING

5.1 Introduction

In active learning the algorithm is given the additional freedom to ask questions about the label of a specific example. This setup captures many natural settings where the algorithm has access to unlabeled data but obtaining labels for the data is expensive. An active learning algorithm can guide the selection of examples that need to be labeled leading to savings in the amount of labeling effort. Since the active learning algorithm effectively compresses the data to a smaller sample, active learning can also save computational effort. In other words, the learning algorithm requires less time because it runs on a smaller input.

At first it seems that enabling the learning algorithm to ask questions should make its task easier: by ignoring redundant examples it should achieve good error rates without asking for too many labels. However, this is not always true. If the space of classifiers that the algorithm is considering does not contain a classifier that labels everything perfectly, then active learning does not necessarily provide an advantage over passive learning [42]. The same is true if the *labeling oracle*, that provides the actual labels to the active learning algorithm, is very noisy. In this chapter we will not make any assumptions about the level of noise in the labels, nor about the best classifier in the algorithm's hypothesis space. Thus the active learning algorithm of this chapter will be *agnostic*.

Agnostic active learning algorithms have been previously proposed [2, 5] but have mostly remained objects of theoretical study. In this chapter, we take a careful look into the details of the recently proposed active learning algorithm

of [5]. The original formulation of this algorithm, though tractable, is highly inefficient. The purpose of this chapter is to show that it is possible to reformulate the algorithm so that it is extremely efficient in the special case of linear classifiers.

The chapter starts by reviewing the algorithm of [5] in Section 5.2. Starting with Section 5.3, the original contributions of this thesis are presented. Namely, an efficient reformulation of the algorithm of [5], as well as an analytically tractable case that further explicates the workings of the algorithm. Finally, we present some empirical results that show the effectiveness of our reformulation.

5.2 Reducing Active to Supervised Learning

We start by describing the algorithm of [5] which is a general purpose reduction from active learning to supervised learning. In other words, this algorithm can be used to turn any supervised learning algorithm into an active learning algorithm. We have chosen this algorithm as a starting point because its design is robust: even if some of the quantities needed by the algorithm are computed approximately, the algorithm will not catastrophically fail.

In contrast with the rest of the algorithms in this thesis, the analysis of active learning assumes that examples actually come from an unknown but fixed distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the input space and $\mathcal{Y} = \{\pm 1\}$ are the labels. Here $(x, y) \in \mathcal{X} \times \mathcal{Y}$ will be a pair of random variables with joint distribution $\mathcal{D}$. An active learner receives a sequence $(x_1, y_1), (x_2, y_2), \ldots$ of i.i.d. copies of (x, y), with the label y_i hidden unless the algorithm explicitly asks for it.

Let $\mathcal{H}$ be a set of hypotheses mapping from $\mathcal{X}$ to $\mathcal{Y}$. We assume that the classifiers in $\mathcal{H}$ do not completely agree on any single $x \in \mathcal{X}$ i.e. $\forall x \in \mathcal{X}, \exists h, h' \in \mathcal{H}$ such that $h(x) \neq h'(x)$. For example, this assumption is true for linear classifiers

that are represented by vectors in a Euclidean ball of radius R, i.e. $\mathcal{H} = \{w : ||w|| \leq R\}$. Then if $h(x) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x})$ we can always use $h'(x) = \text{sign}(-\boldsymbol{w}^\top \boldsymbol{x})$. This is possible because if $\boldsymbol{w} \in \mathcal{H}$, then $-\boldsymbol{w} \in \mathcal{H}$. The error of a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ is $\text{err}(h) := \mathbb{P}(h(x) \neq y)$ where (x, y) is drawn from $\mathcal{D}$. Let $h^* := \arg\min\{\text{err}(h) : h \in \mathcal{H}\}$ be a hypothesis of minimum error in $\mathcal{H}$. The goal of the active learner is to return a hypothesis $h \in \mathcal{H}$ with error $\text{err}(h)$ not much more than $\text{err}(h^*)$, using as few label queries as possible.

In the importance-weighted active learning (IWAL) framework of [4], the active learner looks at the unlabeled data $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ one at a time. Similar to the Label Efficient Perceptron of Section 2.4.1, after each new example $\boldsymbol{x}_t$, the learner determines a probability p_t and a coin with bias p_t is flipped. The label y_t is queried if the coin comes up heads.

To determine the probability p_t, the IWAL framework essentially performs a statistical hypothesis test. Consider two plausible hypotheses that predict differently on the current example and measure the difference in their error rates. If the difference is not too large, then the algorithm cannot deduce the label of the example and should ask for it. If one of the hypotheses is clearly better, then the algorithm can skip this label. However, because we are not making any assumptions on the quality of the previously received labels we hedge our bets by assigning a small probability to the event of querying the oracle. Intuitively, the larger the difference between the error rates of the two hypotheses, the smaller the probability of querying the labeling oracle.

We first turn to the problem of estimating the error rate of a hypothesis h. This task is not trivial because we are interested in the error rate of h on the true distribution of the data while we only have examples that have been collected by the active learning algorithm and potentially constitute a biased sample.

Let $Q_t \in \{0,1\}$ be a random variable that indicates whether the algorithm queries the labeling oracle for label y_t. Formally, Q_t is conditionally independent of the current label y_t, with conditional expectation

$$\mathbb{E}[Q_t | F_{t-1}, \boldsymbol{x}_t] \;=\; p_t$$

where $F_t := \cup_{i=1}^{t} (\boldsymbol{x}_i, y_i, Q_i)$ is the history of the algorithm up to time t.

Given a hypothesis h define its *importance-weighted empirical error* of $\mathrm{err}(h)$ from $F_T \in (\mathcal{X} \times \mathcal{Y} \times \{0,1\})^T$ to be

$$\widehat{\mathrm{err}}(h, F_T) \;:=\; \frac{1}{n} \sum_{t=1}^{T} \frac{Q_t}{p_t} \cdot \mathbb{I}[h(x_t) \neq y_t].$$

Note that this quantity depends on a label y_i only if it has been queried (*i.e.*, only if $Q_i = 1$. In the notation of Algorithm 6, this is equivalent to

$$\mathrm{err}(h, S_T) := \frac{1}{n} \sum_{(x_t, y_t, 1/p_t) \in S_T} \frac{1}{p_t} \cdot \mathbb{I}[h(x_t) \neq y_t] \tag{5.1}$$

where $S_T \subseteq \mathcal{X} \times \mathcal{Y} \times \mathbb{R}$ is the importance weighted sample collected by the algorithm.

A basic property of this estimator is that it has no *bias*:

$$\begin{aligned}
\mathbb{E}[\widehat{\mathrm{err}}(h, F_n)] &= (1/n) \sum_{t=1}^{n} \mathbb{E}[\mathbb{E}[(Q_t/p_t) \cdot \mathbb{I}[h(x_t) \neq y_t] \,|F_{t-1}, x_t, y_t]] \\
&= (1/n) \sum_{i=1}^{n} \mathbb{E}[(p_t/p_t) \cdot \mathbb{I}[h(x_t) \neq y_t] \\
&= \mathbb{E}[\mathbb{I}[h(x_t) \neq y_t] = \mathrm{err}(h).
\end{aligned}$$

This holds for *any* algorithm for setting p_t as long as it guarantees $p_t > 0$.

The full algorithm is shown in Algorithm 6. The rejection threshold (line 8) is based on a deviation bound from [5]. First, the hypothesis h_t, that minimizes the importance weighted error, and the "alternative" hypothesis h'_t are found. Note that both optimizations are over the entire hypothesis class $\mathcal{H}$

Algorithm 6 Importance-weighted active learning with an ERM oracle.

function IWAL(C_0, c_1, c_2 ,T)
  $S_0 \leftarrow \emptyset$
  for $t = 1, 2, \ldots, T$ **do**
    Observe x_t
    $h_t \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \operatorname{err}(h, S_{t-1})$
    $h'_t \leftarrow \operatorname{argmin}_{h \in \mathcal{H} \,\wedge\, h(x_t) \neq h_t(x_t)} \operatorname{err}(h, S_{t-1})$
    $G_t \leftarrow \operatorname{err}(h'_t, S_{t-1}) - \operatorname{err}(h_t, S_{t-1})$
    $p_t \leftarrow \begin{cases} 1 & \text{if } G_t \leq \sqrt{\frac{C_0 \log t}{t-1}} + \frac{C_0 \log t}{t-1} \\ s & \text{otherwise} \end{cases}$ where $0 < s < 1$ satisfies

$$G_t = \left(\frac{c_1}{\sqrt{s}} - c_1 + 1\right) \cdot \sqrt{\frac{C_0 \log t}{t-1}} + \left(\frac{c_2}{s} - c_2 + 1\right) \cdot \frac{C_0 \log t}{t-1}. \tag{5.2}$$

    Draw a Bernoulli random variable Q_t with $\mathbb{P}(Q_t = 1) = p_t$
    if $Q_t = 1$ **then**
      Query the oracle for y_t
      $S_t \leftarrow S_{t-1} \cup \{(x_t, y_t, 1/p_t)\}$
    else
      $S_t \leftarrow S_{t-1}$
    end if
  end for
  return $\operatorname{argmin}_{h \in \mathcal{H}} \operatorname{err}(h, S_T)$
end function

(with h'_t only being required to disagree with h_t on $\boldsymbol{x}_t$)—this is a key aspect where this algorithm differs from previous approaches. The difference in importance-weighted errors G_t of the two hypotheses is then computed. If $G_t \leq \sqrt{(C_0 \log t)/(t-1)} + (C_0 \log t)/(t-1)$, then the query probability p_t is set to 1. Otherwise, p_t is set to the positive solution s to the quadratic equation in (5.2). The functional form of p_t is roughly

$$\min\left\{1,\ \left(\frac{c_2}{G_t} + \frac{2c_1\sqrt{c_2}}{G_t^{3/2}} + O(G_t^{-2})\right) \cdot \frac{C_0 \log t}{t-1}\right\}.$$

The analysis of [5] suggests the following constraints for the constants that appear in the Algorithm: $c_2 \leq c_1$, $c_1 \geq 5 + 2\sqrt{2}$, $c_2 \geq 5$, as well as the value $C_0 = 8$. However, from a practical perspective c_1 and c_2 only have a mild effect on the

behavior of the algorithm as long as they are both greater than 1. The constant C_0 can be thought of as a tuning parameter that describes the labeling cost. If $C_0 \to \infty$ then labels are thought to be very cheap. This makes the algorithm very mellow, meaning that it will ask for all the labels. If $C_0 \to 0$ the labels are very expensive and the algorithm will not ask for any of them.

5.2.1 Consistency

Algorithm 6 enjoys a consistency guarantee that bounds the generalization error of the importance weighted empirical error minimizer. The proof of [5] actually establishes a lower bound on the query probabilities $p_t \geq 1/2$ for $\boldsymbol{x}_t$ such that $h_n(x_t) \neq h^*(x_t)$.

Theorem 9 ([5, Theorem 2]). *The following holds with probability at least* $1 - \delta$*. For any* $n \geq 1$*,*

$$\mathrm{err}(h_n) \;\leq\; \mathrm{err}(h^*) + \sqrt{\frac{2C_0 \log n}{n-1}} + \frac{2C_0 \log n}{n-1}.$$

Therefore, the final hypothesis returned by Algorithm 6 after seeing n unlabeled examples has roughly the same error bound as a hypothesis returned by a standard passive learner with n labeled data.

5.2.2 Label Complexity Analysis

A bound on the number of labels requested by Algorithm 6 after n iterations can also be shown. This is mediated through the *disagreement coefficient*, a quantity first used by [35] for analyzing the label complexity of the A^2 algorithm of [2]. The disagreement coefficient $\theta := \theta(h^*, \mathcal{H}, \mathcal{D})$ is defined as

$$\theta(h^*, \mathcal{H}, \mathcal{D}) := \sup \left\{ \frac{\Pr(X \in \mathrm{DIS}(h^*, r))}{r} : r > 0 \right\}$$

where

$$\mathrm{DIS}(h^*, r) := \{x \in \mathcal{X} : \exists h' \in \mathcal{H} \text{ such that } \Pr(h^*(X) \neq h'(X)) \leq r \text{ and } h^*(x) \neq h'(x)\}$$

(the disagreement region around h^* at radius r). This quantity is bounded for many learning problems studied in the literature; see [35, 36, 31, 64] for more discussion.

Theorem 10 ([5, Theorem 3])**.** *With probability at least $1 - \delta$, the expected number of labels queried by Algorithm 6 after n iterations is at most*

$$1 + \theta \cdot 2\,\mathrm{err}(h^*) \cdot (n-1) + O\left(\theta \cdot \sqrt{C_0 n \log n} + \theta \cdot C_0 \log^3 n\right).$$

The linear term $\mathrm{err}(h^*) \cdot n$ is unavoidable in the worst case, as evident from label complexity lower bounds [42, 4]. When $\mathrm{err}(h^*)$ is negligible and θ is bounded (as is the case for many problems studied in the literature [35]), then the bound represents a polynomial improvement in label complexity over supervised learning.

5.3 Efficient Implementation

The reduction of [5] requires for each unlabeled example two hypotheses:

1. an empirical risk minimizer (ERM)
2. an alternative risk minimizer that disagrees with the ERM on one example.

For all but the simplest hypothesis spaces, finding the ERM is hard. Even for linear classifiers, finding whether the ERM hypothesis makes fewer than k errors is NP-complete. Hence the first approximation is to relax the requirement

to return the ERM hypothesis. Instead of the ERM hypothesis one can use the output of a learning algorithm. For example, a linear support vector machine returns a linear classifier that minimizes a convex upper bound of the empirical risk. Many learning algorithms work in a similar way.

The second approximation involves the alternative hypothesis required by the algorithm. For many popular linear classifiers, such as linear support vector machines or linear logistic regression, finding the alternative hypothesis can be formulated as a convex optimization problem with an additional linear inequality constraint. This constraint simply enforces that the current unlabeled example should have a label different than the one preferred by the ERM hypothesis. This label will be referred to as the *alternative label* and denoted by y_a. For example, for a linear classifier, and assuming that the alternative label is negative ($y_a = -1$), the added constraint is $\boldsymbol{w}^\top \boldsymbol{x}_t \leq 0$. This means that the alternative hypothesis should classify the current example as negative.

For multiclass problems with K classes, finding which of the labels should play the role of the alternative label requires searching over the possible $K - 1$ labels. When testing if the class label k should be the alternative one, constraints are introduced so that example x_t is constrained to belong to class k. For a linear classifier that maintains one weight vector per class and predicts via the rule

$$\hat{y}_t = \operatorname*{argmax}_{1 \leq j \leq K} \boldsymbol{w}_j^\top \boldsymbol{x}_t,$$

we introduce $K - 1$ constraints of the form

$$\boldsymbol{w}_k^\top \boldsymbol{x}_t \geq \boldsymbol{w}_j^\top \boldsymbol{x}_t$$

for all $j \neq k$. The alternative label is then the one that gives the hypothesis with the fewest errors on the training set, with ties broken arbitrarily. Multiclass problems will not be discussed any further in this chapter.

Though the approach with the additional linear constraint is tractable, it requires solving a convex problem. A much simpler way of obtaining a similar result is as follows. Execute the learning algorithm on the collected dataset plus the unlabeled example x_t with the alternative label. Furthermore, the importance of x_t should be just large enough so that the resulting hypothesis predicts the alternative label. This way of obtaining an alternative hypothesis works with any black box learning algorithm able to handle examples with varying importances. If the algorithm is really a black box, binary search can be used to estimate the right importance. However, as discussed below, for linear classifiers this importance can be computed much more cheaply.

The benefit of thinking in terms of importances is in the relation of the importance with the quantity G_t in the active learning algorithm. Recall that G_t is the difference in importance-weighted error rates between the ERM and the alternative hypothesis. In fact, G_t is an estimator of the difference between the true error rates. Therefore, $t \cdot G_t$ is an estimate of the additional number of mistakes that the alternative hypothesis will do on the first t examples. On the other hand, suppose that i_t is the smallest importance that gives a valid alternative hypothesis, i.e. an i' such that $h'(x_t)$ is the alternative label. Then, i_t is an estimate of the additional number of mistakes that the alternative hypothesis will do on the first t examples. To see this, consider a learning algorithm that has collected a random sample S_{t-1} plus i_t copies of x_t with the alternative label. The ERM for this learner is the alternative hypothesis h'. Now present the current example x_t. Clearly this will cause h' to make a mistake. Once the learner receives the correct label, it is convenient to imagine that this cancels one of the i_t copies of x_t that the algorithm started with. Repeating this process i_t times the learner is left with the initial set S_{t-1} whose ERM is the original empirical risk minimizer

h considered by the active learner at step t. Therefore, h and h' were i_t mistakes apart. Thus i_t will be used to estimate $t \cdot G_t$, or $G_t = i_t/t$. In conclusion, to run Algorithm 6, the alternative hypothesis is not necessary. One only needs G_t, or a reasonable way to estimate it. Recall that Algorithm 6 is consistent and hence robust to the kinds of approximations discussed here.

The next observation is that calling a learning algorithm on every iteration to compute h seems very wasteful. Clearly, the dataset on which the algorithm is executed differs from the previous one by just one example. For this case, many specialized algorithms exist, from Bayesian updating such as [40] to warm-starting an optimization method [12]. The approach taken here is even more radical: each new example is handed over to an online learning algorithm. Thus it will be processed once and will never be revisited. Since the active learning algorithm assigns importances to the examples, the variants of online gradient descent described in Chapter 4 will be used. In other words, the weight vector $\boldsymbol{w}_t$ at each iteration of online gradient descent will serve as the ERM hypothesis for that round.

Finally, using the importance invariant updates, one can obtain a closed form solution for the importance that should be given to the current unlabeled example.

Theorem 11. *The importance i_t that will cause the next iterate w_{t+1} to classify x_t as $y_a = -\operatorname{sign}(w_t^\top x_t)$ after an invariant gradient update is*

$$i_t = \frac{1}{\eta_t} \int_0^{\frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{||\boldsymbol{x}_t||^2}} \left(\left. \frac{\partial \ell}{\partial p} \right|_{p=(\boldsymbol{w}_t - u\boldsymbol{x}_t)^\top \boldsymbol{x}_t} \right)^{-1} du, \tag{5.3}$$

where $\ell = \ell(p, y_a)$ is the loss of predicting p when the actual prediction is y_a.

Proof. The importance should be just large enough that an invariant update with the triple $(\boldsymbol{x}_t, y_a, i_t)$ leads to a hypothesis $\boldsymbol{w}'$ such that $\operatorname{sign}(\boldsymbol{w}'^\top \boldsymbol{x}_t) = y_a$. Here

$y_a = -\operatorname{sign}(\boldsymbol{w}_t^\top x_t)$. Recall that the invariant update in this case is

$$\boldsymbol{w}' = \boldsymbol{w}_t - s(i_t)\boldsymbol{x}_t, \tag{5.4}$$

where s satisfies the ODE

$$\frac{\partial s}{\partial i} = \eta_t \left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - s(i)\boldsymbol{x})^\top \boldsymbol{x}} \qquad s(0) = 0. \tag{5.5}$$

Notice that now i_t is sought so an explicit solution of the ODE in terms of s is not necessary. In fact it is sufficient that $\boldsymbol{w}'^\top \boldsymbol{x}_t = 0$, so using (5.4)

$$\boldsymbol{w}_t^\top \boldsymbol{x}_t - s(i_t)||\boldsymbol{x}_t||^2 = 0,$$

and hence

$$s(i_t) = \frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{||\boldsymbol{x}_t||^2}. \tag{5.6}$$

Going back to (5.5), separating variables, integrating both sides, and using the initial condition one gets that

$$i_t = \frac{1}{\eta_t} \int_0^{\frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{||\boldsymbol{x}_t||^2}} \left(\left.\frac{\partial \ell}{\partial p}\right|_{p=(\boldsymbol{w}_t - u\boldsymbol{x}_t)^\top \boldsymbol{x}_t} \right)^{-1} du.$$

□

The following corollaries are direct consequences:

Corollary 1. *For the logistic loss* $\ell(p, y) = \log(1 + \exp(-yp))$,

$$i_t = \frac{1 - y_a \boldsymbol{w}_t^\top \boldsymbol{x}_t - \exp(y_a \boldsymbol{w}_t^\top \boldsymbol{x}_t)}{\eta_t ||\boldsymbol{x}_t||^2}.$$

Corollary 2. *For the squared loss* $\ell(p, y) = \frac{1}{2}(p - y)^2$, *assuming* $y \in \{-1, +1\}$ *and* $-1 < \boldsymbol{w}_t^\top \boldsymbol{x}_t < 1$

$$i_t = \frac{\log(1 - y_a \boldsymbol{w}_t^\top \boldsymbol{x}_t)}{\eta_t ||\boldsymbol{x}_t||^2}.$$

It is also possible to find i_t when implicit updates are used but there is no general methodology. However, for many loss functions it is possible to find a closed form solution for the importance. Below we show how to do this in the case of logistic loss.

For the logistic loss the optimization problem that defines the implicit update is

$$\boldsymbol{w}_{t+1} = \operatorname{argmin} \frac{1}{2}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + i_t \eta_t \log(1 + \exp(-y_a \boldsymbol{w}^\top \boldsymbol{x}_t)).$$

Setting the derivative of the above to zero leads to

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{i_t \eta_t y_a}{1 + \exp(y_a \boldsymbol{w}_{t+1}^\top \boldsymbol{x}_t)} \boldsymbol{x}_t. \tag{5.7}$$

This update is called implicit because w_{t+1} is given in terms of itself. Nevertheless, there is a unique w_{t+1} that satisfies (5.7) and can be found with a root-finding procedure. In our case we do not even need w_{t+1}, but only the importance i_t. To do that, it is sufficient to require $\boldsymbol{w}^\top \boldsymbol{x}_t = 0$ and by taking the inner product of (5.7) with $\boldsymbol{x}_t$ and plugging $\boldsymbol{w}_{t+1}^\top \boldsymbol{x}_t = 0$ in we get

$$0 = \boldsymbol{w}_t^\top \boldsymbol{x}_t + \frac{i_t \eta_t y_a}{2} \boldsymbol{x}_t^\top \boldsymbol{x}_t$$

$$i_t = -\frac{2 \boldsymbol{w}_t^\top \boldsymbol{x}_t}{\eta_t y_a \boldsymbol{x}_t^\top \boldsymbol{x}_t}$$

The complete algorithm is shown in Algorithm 7. The algorithm provides explicit and numerically stable formulas for the probabilities p_t using $c_1 = c_2 = 2$ for the constants of Algorithm 6. Of particular interest is the dual role that importances play in this algorithm. On one hand, the importance i_t helps determine the difference between the empirically best and the alternative hypotheses. On the other hand, the quantity $1/p_t$ is used as the importance of $\boldsymbol{x}_t$ to keep the sample S_t unbiased.

Algorithm 7 Importance weighted active learning with online gradient

function FAST-IWAL(C_0 ,T)
  $S_1 \leftarrow \emptyset$
  $\boldsymbol{w}_1 = \mathbf{0}$
  for $t = 1, 2, \ldots, T$ **do**
    Observe $\boldsymbol{x}_t$
    $y_a \leftarrow -\operatorname{sign}(\boldsymbol{w}_t^\top \boldsymbol{x}_t)$
    $i_t \leftarrow {\eta_t}^{-1} \int_0^{\frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{||\boldsymbol{x}_t||^2}} \left(\frac{\partial \ell(p, y_a)}{\partial p}\Big|_{p=(\boldsymbol{w}_t - u\boldsymbol{x}_t)^\top \boldsymbol{x}_t} \right)^{-1} du$
    $G_t \leftarrow i_t / t$
    $b_t \leftarrow \sqrt{\frac{C_0 \log t}{t-1}}$
    $p_t \leftarrow \begin{cases} 1 & \text{if } G_t \leq b_t + b_t^2 \\ \left(\frac{b_t \left(1 + \sqrt{1 + 2(G_t + b_t^2 + b_t)}\right)}{G_t + b_t^2 + b_t} \right)^2 & \text{otherwise} \end{cases}$
    Draw a Bernoulli random variable Q_t with $\mathbb{P}(Q_t = 1) = p_t$
    if $Q_t = 1$ **then**
      Query the oracle for y_t
      $S_{t+1} \leftarrow S_t \cup \{(\boldsymbol{x}_t, y_t, 1/p_t)\}$
      $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - s(1/p_t)\boldsymbol{x}_t$
    else
      $S_{t+1} \leftarrow S_t$
      $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t$
    end if
  end for
  return w_{T+1}, S_{T+1}
end function

5.4 An Analytically Tractable Case

Besides the approximations presented so far, there is also a special case for which a quantity similar to G_t can be computed in closed form. Suppose that at time t a set of examples S_t has been collected, and let Δ_t be the difference in importance-weighted loss between the current classifier and the alternative one, i.e.

$$\Delta_t = \sum_{(\boldsymbol{x}, y, i) \in S_t} i(\ell(\boldsymbol{w}_t'^\top \boldsymbol{x}, y) - \ell(\boldsymbol{w}_t^\top \boldsymbol{x}, y)).$$

Furthermore, suppose w_t is learned using *least squares classification*, meaning that

$$\boldsymbol{w}_t = \operatorname*{argmin}_{w} \sum_{(\boldsymbol{x},y,i)\in S_t} \frac{i}{2}(\boldsymbol{w}^\top \boldsymbol{x} - y)^2,$$

where $y \in \{-1, +1\}$. This can be computed in closed form as

$$\boldsymbol{w}_t = \boldsymbol{A}_t^{-1}\boldsymbol{q}_t \tag{5.8}$$

where

$$\boldsymbol{A}_t = \sum_{(x,y,i)\in S_t} i\boldsymbol{x}\boldsymbol{x}^\top \tag{5.9}$$

$$\boldsymbol{q}_t = \sum_{(\boldsymbol{x},y,i)\in S_t} iy\boldsymbol{x}. \tag{5.10}$$

For now, we assume $\boldsymbol{A}_t$ is invertible and we discuss a related setting where $\boldsymbol{A}_t$ is guaranteed to be so at the end of this section.

In general, least squares classification is not a good idea. For example, when $y = 1$ a prediction $\boldsymbol{w}^\top \boldsymbol{x} = 0$ attains the same loss as prediction $\boldsymbol{w}^\top \boldsymbol{x} = 2$. However, the former gives no indication of the actual label while the latter prediction suggests that $\boldsymbol{x}$ might be positive. Regardless of its suitability for classification purposes, squared loss provides closed-form solutions from which useful insights can be obtained. This is illustrated in the following theorem:

Theorem 12. *For a classifier learned with least squares classification, we have*

$$\Delta_t = \frac{(\boldsymbol{w}_t^\top \boldsymbol{x}_t)^2}{2\boldsymbol{x}_t^\top \boldsymbol{A}_t^{-1}\boldsymbol{x}_t}, \tag{5.11}$$

where $\boldsymbol{w}_t$ and $\boldsymbol{A}_t$ are defined in (5.8) *and* (5.9) *respectively.*

Proof. The alternative classifier is the minimizer of

$$\begin{aligned} \boldsymbol{w}_t' &= \operatorname*{argmin}_{w} \sum_{(\boldsymbol{x},y,i)\in S_t} \frac{i}{2}(\boldsymbol{w}^\top \boldsymbol{x} - y)^2 \\ \text{subject to} \quad & \boldsymbol{w}^\top \boldsymbol{x}_t = 0, \end{aligned}$$

The Lagrangian of this problem is

$$L(w, \lambda) = \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{(x,y,i)\in S_t} \frac{i}{2}(\boldsymbol{w}^\top x - y)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{x}_t.$$

Setting the partial derivative of L with respect to $\boldsymbol{w}$ to zero yields

$$\boldsymbol{w} = \boldsymbol{A}_t^{-1}(\boldsymbol{q}_t - \lambda \boldsymbol{x}_t).$$

We now plug this back in the constraint $\boldsymbol{w}^\top \boldsymbol{x}_t = 0$ and solve for λ, yielding

$$\lambda = \frac{\boldsymbol{x}_t^\top \boldsymbol{A}_t^{-1} \boldsymbol{q}_t}{\boldsymbol{x}_t \boldsymbol{A}_t^{-1} \boldsymbol{x}_t},$$

therefore

$$\boldsymbol{w}_t' = \boldsymbol{A}_t^{-1}\boldsymbol{q}_t - \frac{\boldsymbol{x}_t^\top \boldsymbol{A}_t^{-1} \boldsymbol{q}_t}{\boldsymbol{x}_t \boldsymbol{A}_t^{-1} \boldsymbol{x}_t} \boldsymbol{A}_t^{-1} \boldsymbol{x}_t. \tag{5.12}$$

On the other hand, Δ_t can be expressed succinctly as

$$\Delta_t = \frac{1}{2}\boldsymbol{w}_t'^\top \boldsymbol{A}_t \boldsymbol{w}_t' - \boldsymbol{q}_t^\top \boldsymbol{w}_t' - \frac{1}{2}\boldsymbol{w}_t^\top \boldsymbol{A}_t \boldsymbol{w}_t + \boldsymbol{q}_t^\top \boldsymbol{w}_t.$$

Substituting the values of $\boldsymbol{w}_t$ and $\boldsymbol{w}_t'$ from (5.8) and (5.12) leads to

$$\Delta_t = \frac{(\boldsymbol{x}_t^\top \boldsymbol{A}_t^{-1} \boldsymbol{q}_t)^2}{\boldsymbol{x}_t^\top \boldsymbol{A}_t^{-1} \boldsymbol{x}_t} = \frac{(\boldsymbol{w}_t^\top \boldsymbol{x}_t)^2}{2\boldsymbol{x}_t^\top \boldsymbol{A}_t^{-1} \boldsymbol{x}_t}.$$

□

The probability of asking for a label will be large when Δ_t is small and Theorem 12 elucidates when this is true depending on $\boldsymbol{w}_t$, $\boldsymbol{x}_t$, and S_t. On one hand, if the magnitude of the current prediction $\boldsymbol{w}_t^\top \boldsymbol{x}_t$ is close to 0, then the classifier is uncertain about its prediction and it makes sense to ask for the label. On the other hand, Δ_t depends on $\boldsymbol{x}_t \boldsymbol{A}_t^{-1} \boldsymbol{x}_t$ which is the norm of $\boldsymbol{x}_t$ measured according to a metric defined by the examples in S_t. If $\boldsymbol{x}_t$ is very different from the examples in S_t, then $\boldsymbol{x}_t \boldsymbol{A}_t^{-1} \boldsymbol{x}_t$ will be large and Δ_t will be small. Again, it is sensible to ask for the label in this case. Interestingly, the same quantity as Δ_t

appears in the querying criterion of another recently proposed active learning algorithm [25] that works only for least squares classification.

Finally, to address any concerns of singular or ill-conditioned $\boldsymbol{A}_t$, one can solve a regularized problem:

$$\boldsymbol{w}_t = \operatorname*{argmin}_w \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{(\boldsymbol{x},y,i)\in S_t} \frac{i}{2}(\boldsymbol{w}^\top \boldsymbol{x} - y)^2.$$

In this case, the expression for Δ_t is only slightly more complicated:

$$\Delta_t = \frac{(\boldsymbol{w}_t^\top \boldsymbol{x}_t)^2}{2\boldsymbol{x}_t^\top \boldsymbol{C}_t^{-1}\boldsymbol{x}_t} + \frac{1}{2}(||\boldsymbol{w}_t||^2 - ||\boldsymbol{w}'_t||^2),$$

where $\boldsymbol{C}_t$ is defined as $\boldsymbol{C}_t = \boldsymbol{I} + \sum_{(x,y,i)\in S_t} i\boldsymbol{x}\boldsymbol{x}^\top$, while $\boldsymbol{w}_t$ and $\boldsymbol{w}'_t$ have been redefined as

$$\boldsymbol{w}_t = \boldsymbol{C}_t^{-1}\boldsymbol{q}_t$$

$$\boldsymbol{w}'_t = \boldsymbol{w}_t - \frac{\boldsymbol{w}_t^\top \boldsymbol{x}_t}{\boldsymbol{x}_t^\top \boldsymbol{C}_t^{-1}\boldsymbol{x}_t}\boldsymbol{C}_t^{-1}\boldsymbol{x}_t.$$

5.5 Experiments

To experimentally evaluate Algorithm 7, the four datasets used in Chapter 4 were treated as active learning tasks. The learning rates were set as in the experiments of Chapter 4 and again only one pass through the training set was performed. The algorithm uses a linear model, with a link function $\sigma(p) = \max(-1, \min(1, p))$, and optimizes squared loss as implemented in the Vowpal Wabbit [48] system.

Some crude calculations taking into account the form of the probability p_t and the dependence of G_t on η_t and t show that when η_t decays as $1/t$ then the generated importance weights $1/p_t$ scale like $O(t/\log(t))$, so they are expected to become large as the algorithm runs. Hence it is important to be able to handle these weights correctly.

As a side note, the probabilities themselves become small, in this case $O(\log(t)/t)$. Summing over all t, the expected number of collected labels is $O(\log^2(t))$. The good news here is that the number of collected labels grows slowly and remains unbounded, so the algorithm will eventually collect enough labels to learn. The bad news is that this result is not what Theorem 10 predicts. Perhaps the analysis here is not as refined as necessary, or perhaps a learning rate of $1/t$ is not an adequate choice. On the other hand, the same crude calculations with a learning rate of $1/\sqrt{t}$ suggest that $1/p_t$ stays bounded regardless of the value of t. However, this is not validated by our experimental results where we see that careful handling of large importances is crucial.

In the experiments, three update strategies were used to handle the generated importance weights $1/p_t$. The first was the naive approach of multiplying the gradient with the importance weight, the second strategy used the invariant updates of Section 4.3, and the third strategy used the implicit updates of Section 4.5. Algorithm 7 was also modified so that the importance weight i_t was computed according to each strategy. In other words, i_t is such that if the algorithm were to update with $(\boldsymbol{x}_t, y_a, i_t)$ then the resulting classifier would have predicted y_a as the label of $\boldsymbol{x}_t$.

In Figures 5.1 and 5.2 we summarize our results. Each combination of learning rate schedule and setting of the parameter C_0 in Algorithm 7 ($C_0 \in \{10^{-8}, 10^{-7}, \ldots, 10^{1}\}$) is an experiment that can be represented in the graph by a point whose x-coordinate is the fraction of labels queried by the active learning algorithm and whose y-coordinate is the test error of the learned hypothesis. To summarize this set of points, the figures plot part of its convex hull. The points on the convex hull (sometimes called a Pareto frontier) are experiments which represent optimal tradeoffs between generalization and label complex-

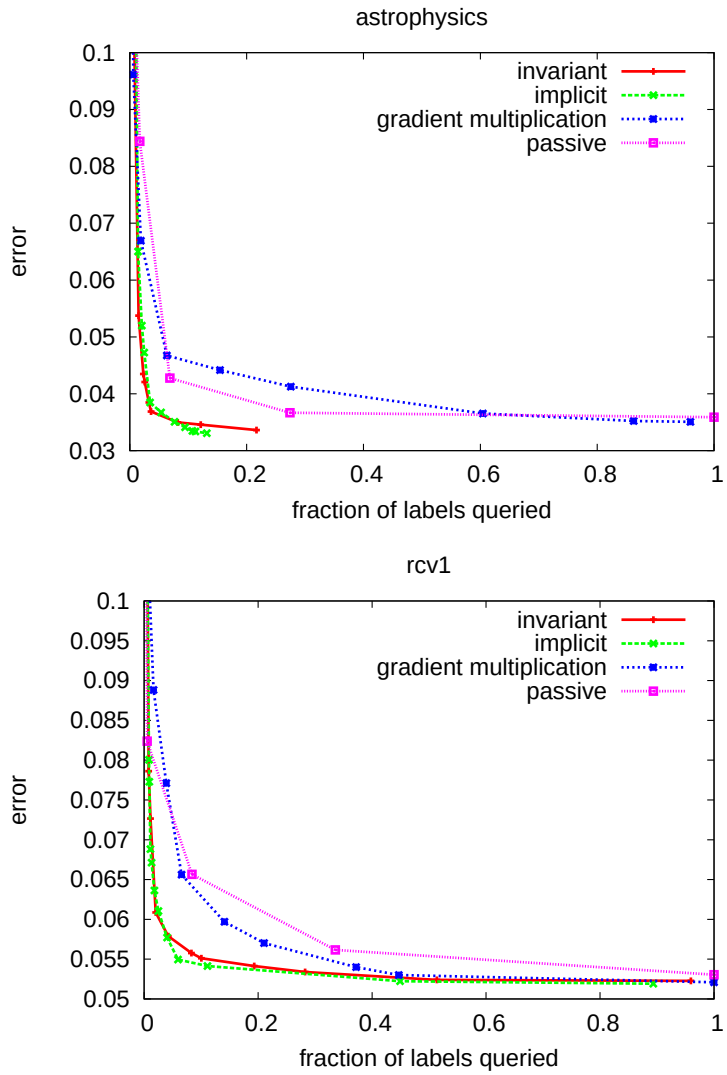

Figure 5.1: Test errors as a function of the number of labels queried for online learning experiments.

ity for some setting of this tradeoff. When a curve stops sooner than the size of the dataset, it means that there were no experiments in which using more queries gave better generalization. We have also included the results from a typical good run of a passive learner. The graphs show very convincingly the value of having an update that handles importance weights correctly. Doing so yields better generalization and lower label complexity than those attainable by

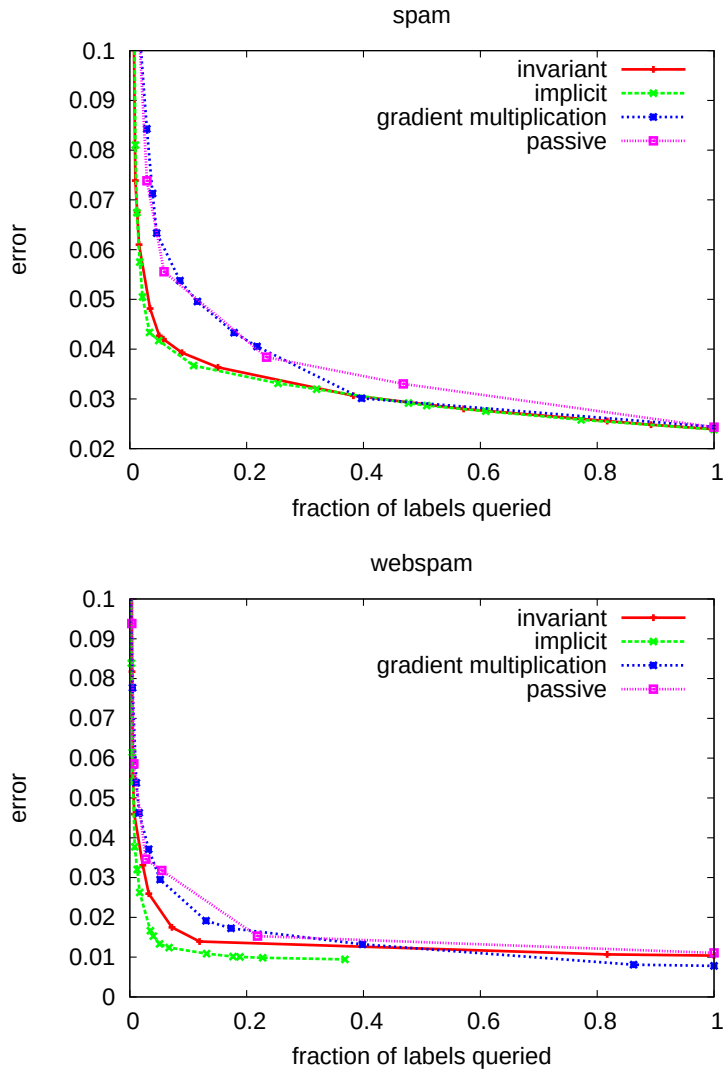

Figure 5.2: Test errors as a function of the number of labels queried for online learning experiments.

multiplying the gradient with the importance weight. In fact, Table 5.1 lists the ratio of labels between passive and active learning to achieve a given accuracy. This shows that naive handling of importances can make active learning need more labels than passive learning to achieve the same accuracy.

Table 5.1: Reduction in label complexity

Dataset	astro	rcv1	spam	web
Desired Accuracy	0.963	0.943	0.967	0.986
Multiplication	0.45	1.59	1.28	0.54
Implicit	5.12	6.55	1.88	4.33
Invariant	7.56	6.55	2.13	1.82

5.6 Conclusion

This chapter provided an efficient implementation of the algorithm in [5] which is as fast as online learning, is theoretically sound, and produces a dataset that can be reused for learning other classifiers with perhaps different representations. It also demonstrated the empirical effectiveness of this algorithm in four different problems. Even though many of the computational shortcuts taken were approximate or heuristic, the created active learning algorithm never failed and in fact it would eventually converge to the same solution as a passive supervised learning algorithm. Consequently, this approach can be widely used to reduce the cost of labeling in situations where labeling is expensive.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

This thesis has presented solutions for various online learning tasks by building on top of existing algorithms. The algorithm of Chapter 3 is a modification of Winnow and ensures that the learned classifier uses little memory. The updates of Chapter 4 are the result of online gradient descent in continuous time, and the fast active learning algorithm of Chapter 5 is based on a slow algorithm together with creative uses for the importances of the examples.

In all cases, the insights that led to the improvements came after careful examination of the existing algorithms. For the PST algorithm of Chapter 4 it was a combination of two key ideas. First, realizing that the optimal tree should be sparse; and second, that the damage caused by approximate updates is minimized when a multiplicative update algorithm is used. For the careful treatment of importance weights, the insight came from understanding that online gradient descent implicitly treats the loss function as linear. Finally, the fast active learning algorithm of Chapter 5 is based on approximations that obviate the need to compute the alternative hypothesis $\boldsymbol{w}'_t$.

In fact, online algorithms are simple algorithms and the proofs of their guarantees are transparent. Every time the online algorithm makes an update, it moves closer to the optimal hypothesis. This suggests that online learning algorithms can be easily adapted for many other tasks. To do this, one first has to understand how various externally imposed requirements can be incorporated into the algorithm. For example, as this thesis showed, a small model can be obtained by approximate updates. In this case, keeping track of the effect of the approximation in the proof suggests that certain terms should be bounded in order to maintain a nontrivial mistake bound for the whole algorithm. In general,

proofs can guide the design of a learning algorithm to ensure that the algorithm still entertains some guarantees when a desired requirement is imposed.

Even though online learning algorithms are very robust and come with good guarantees, in practice they often produce classifiers whose error rate is slightly worse than a classifier obtained via a batch learning algorithm. This is especially true for datasets that can fit in the main memory of a single machine. The remedy in such cases is to cycle the data through the online algorithm more than once. When do we stop training? Many machine learning tasks are convex, hence it is possible to estimate the suboptimality of a candidate solution using convex duality. Such an estimate of an upper bound on the duality gap is in sharp contrast to a typical approach that would measure convergence on a set of held out data; here no data needs to be held out. Furthermore, we can compute this surrogate duality gap in an online manner as we run the algorithm. For very large problems, such a stopping criterion opens the possibility of learning without even going through the whole input.

For active learning, the algorithms need further refinements to be practically useful. On one hand, extensions to multiclass problems need to be investigated. On the other hand, settings with more constraints need to be addressed. For example, it is often the case that acquiring labels in bulk is cheaper than acquiring labels one by one. This leads to the *batch mode active learning* setting [34] where the algorithm is given a pool of unlabeled examples and has to select k at a time to ask for their labels. Clearly, one can compute a probability similar to that of Algorithm 7 for each unlabeled example in the pool. However, picking the k examples based on these probabilities might not be a great idea. For example, two similar unlabeled examples will both be equally good according to these probabilities. However, once we know the label for one of them, the other

might not be very informative. Therefore, for batch mode active learning some issues regarding the diversity of the k queried examples need to be addressed. Moreover, the possible labelings of the k queried examples are 2^k for binary classification, and reasoning about exponentially many possibilities constitutes an additional difficulty in this setting.

Another common shortcoming of the online algorithms in this thesis and those most analyzed in the literature is that they maintain only one hypothesis at a time. From an engineering point of view, the bottleneck in online learning algorithms is reading the data. Hence maintaining a small number of hypotheses, say 20, will not cause any degradation in performance. Having more than one hypothesis can be useful for tracking a constantly drifting concept. This might be the case when the learning algorithm must recommend current events, where new trends and stories quickly appear and disappear. In such a case, different hypotheses can be learned for different time resolutions and with different emphasis on recency. Another possibility is to obtain a measure of uncertainty for a prediction. For example, such functionality can be used to decide whether it is worth asking for the label of an example in active learning. Suppose that some hypotheses disagree on the classification of the example. Then asking for the label is going to significantly update at least one of these hypotheses. However, if all hypotheses agree, one might still want to ask for the label in case all of the hypotheses are mistaken at the same time. A simple way to introduce diversity among the hypotheses is via randomization of the importance of each example. For example, the importances generated by *Bootstrapping* [29], a statistical resampling method, are outcomes of a binomial random variable. This random variable describes the number of successes in n trials with each trial having success probability i/n, where i is the actual importance and n is

the sum of importances over all examples. When n is large, the distribution of this random variable is well approximated by a Poisson distribution with rate i. This ensures that in expectation every example appears in the data with its original importance.

Finally, all the algorithms in this thesis work in the setting of full feedback. However, in many scenarios the algorithm makes a prediction and only observes whether a human agrees with that prediction. This is true in most settings in the World Wide Web, where algorithms that recommend movies, ads, or search results never explicitly learn what the user would have liked to see instead. In order for the algorithm to learn in such a setting, it has to intelligently collect enough information about its alternatives. This is called *exploration*. Clearly the algorithm should often suggest the best alternative given the collected information. This is called *exploitation*. Balancing the tradeoff between exploration and exploitation in online learning is a problem that has received much attention. Solutions exist for restricted settings and heuristics are already employed in search engines to recommend ads [33]. However, an algorithm that is general, efficient, and comes with a theoretical guarantee has remained elusive. One line of future work is to better understand the heuristics [33, 14]. Another possibility is to start with a theoretically sound algorithm. In [28] we have shown an algorithm with a theoretical guarantee that makes polynomially many calls to a learning algorithm in order to make a single prediction. Making such an algorithm practical might be possible by considering simple modifications and perhaps sacrificing its optimal $O(\sqrt{T})$ regret.

BIBLIOGRAPHY

[1] A. Agarwal, P.L. Bartlett, P. Ravikumar, and M.J. Wainwright. Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization. *Information Theory, IEEE Transactions on*, (99), 2010.

[2] M.-F. Balcan, A. Beygelzimer, and J. Langford. Agnostic active learning. In *Twenty-Third International Conference on Machine Learning*, 2006.

[3] M.-F. Balcan, A. Broder, and T. Zhang. Margin based active learning. In *Twentieth Annual Conference on Learning Theory*, 2007.

[4] A. Beygelzimer, S. Dasgupta, and J. Langford. Importance weighted active learning. In *Twenty-Sixth International Conference on Machine Learning*, 2009.

[5] A. Beygelzimer, D. Hsu, J. Langford, and T. Zhang. Agnostic active learning without constraints. In *Advances in Neural Information Processing Systems 23*, 2010.

[6] Alina Beygelzimer, John Langford, and Bianca Zadrozny. Machine learning techniques—reductions between prediction quality metrics. In Z. Liu and C. H. Xia, editors, *Performance Modeling and Engineering*, pages 3–28. Springer-Verlag, 2008.

[7] A. Blum. Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning*, 26(1):5–23, 1997.

[8] A. Blum, A. Kalai, and J. Langford. Beating the holdout: Bounds for kfold and progressive cross-validation. In *COLT*, 1999.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[10] H. Brendan McMahan and M. Streeter. Adaptive Bound Optimization for Online Convex Optimization. *COLT*, 2010.

[11] P. Buhlmann and A.J. Wyner. Variable length Markov chains. *Annals of Statistics*, 27(2):480–513, 1999.

[12] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *NIPS*, pages 409–415, 2000.

[13] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[14] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *NIPS*, pages 2249–2257, 2011.

[15] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[16] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

[17] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[18] A. Culotta and A. McCallum. Reducing labeling effort for structured prediction tasks. In *Proceedings Of The National Conference On Artificial Intelligence*, volume 20, page 746, 2005.

[19] Ido Dagan and Sean P. Engelson. Committee-based sampling for training probabilistic classifiers. In *ICML*, pages 150–157, 1995.

[20] S. Dasgupta. Coarse sample complexity bounds for active learning. In *Advances in Neural Information Processing Systems 18*, 2005.

[21] S. Dasgupta, D. Hsu, and C. Monteleoni. A general agnostic active learning algorithm. In *Advances in Neural Information Processing Systems 20*, 2007.

[22] Sanjoy Dasgupta. Two faces of active learning. *Theor. Comput. Sci.*, 412(19):1767–1781, 2011.

[23] O. Dekel, S. Shalev-Shwartz, and Y. Singer. The power of selective memory: Self-bounded learning of prediction suffix trees. *Advances in Neural Information Processing Systems*, 17, 2004.

[24] O. Dekel, S. Shalev-Shwartz, and Y. Singer. Smooth ε-Insensitive Regression by Loss Symmetrization. *Journal of Machine Learning Research*, 6:711–741, 2005.

[25] Ofer Dekel, Claudio Gentile, and Karthik Sridharan. Robust selective sampling from single and multiple teachers. In *COLT*, pages 346–358, 2010.

[26] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *COLT*, 2010.

[27] J. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. *The Journal of Machine Learning Research*, 10:2899–2934, 2009.

[28] Miroslav Dudík, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. Efficient optimal learning for contextual bandits. In *UAI*, pages 169–178, 2011.

[29] B. Efron and R. Tibshirani. *An introduction to the bootstrap*, volume 57. Chapman & Hall/CRC, 1993.

[30] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*. Springer, 1995.

[31] E. Friedman. Active learning for smooth problems. In *Twenty-Second Annual Conference on Learning Theory*, 2009.

[32] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. WH Freeman & Co, 1979.

[33] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *ICML*, pages 13–20, 2010.

[34] Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In *NIPS*, 2007.

[35] S. Hanneke. A bound on the label complexity of agnostic active learning. In *Twenty-Fourth International Conference on Machine Learning*, 2007.

[36] S. Hanneke. Adaptive rates of convergence in active learning. In *Twenty-Second Annual Conference on Learning Theory*, 2009.

[37] D.P. Helmbold and R.E. Schapire. Predicting Nearly As Well As the Best Pruning of a Decision Tree. *Machine Learning*, 27(1):51–68, 1997.

[38] M. Herbster. Learning additive models online with fast evaluating kernels. In *Computational Learning Theory*. Springer, 2001.

[39] J. Huang, A.J. Smola, A. Gretton, K.M. Borgwardt, and B. Scholkopf. Correcting sample selection bias by unlabeled data. *Advances in neural information processing systems*, 19, 2007.

[40] T.S. Jaakkola and M.I. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10(1):25–37, 2000.

[41] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006.

[42] M. Kääriäinen. Active learning in the non-realizable case. In *Seventeenth International Conference on Algorithmic Learning Theory*, 2006.

[43] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[44] M. Kearns and Y. Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 269–277, 1998.

[45] J. Kivinen and M.K. Warmuth. Exponentiated Gradient versus Gradient Descent for Linear Predictors. *Informatics and Computation*, 132:1–64, 1997.

[46] V. Koltchinskii. Rademacher complexities and bounding the excess risk in active learning. *Journal of Machine Learning Research*, 11:2457–2485, 2010.

[47] B. Kulis and P. Bartlett. Implicit Online Learning. In *ICML 2010*, 2010.

[48] J. Langford, L. Li, and A. Strehl. Vowpal wabbit online learning project, 2007. `http://hunch.net/?p=309`.

[49] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *The Journal of Machine Learning Research*, 10:777–801, 2009.

[50] D.D. Lewis and W.A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[51] D.D. Lewis, Y. Yang, T.G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[52] N. Littlestone. *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, Santa Cruz, CA, USA, 1989.

[53] M. Marden. *Geometry of Polynomials*. Amer Mathematical Society, 1966.

[54] Albert B. Novikoff. On convergence proofs for perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1963.

[55] F. Pereira and Y. Singer. An Efficient Extension to Mixture Techniques for Prediction and Decision Trees. *Machine Learning*, 36(3):183–199, 1999.

[56] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making stochastic gradient descent optimal for strongly convex problems. In *ICML 2012*, 2012.

[57] D. Ron, Y. Singer, and N. Tishby. The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning*, 25(2):117–149, 1996.

[58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[59] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[60] T. Scheffer, C. Decomain, and S. Wrobel. Active hidden markov models for information extraction. *Advances in Intelligent Data Analysis*, pages 309–318, 2001.

[61] S. Shalev-Shwartz. *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, The Hebrew University of Jerusalem, 2007.

[62] S. Shalev-Shwartz and Y. Singer. Logarithmic regret algorithms for strongly convex repeated games. *The Hebrew University, Technical Report*, 2007.

[63] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[64] L. Wang. Sufficient conditions for agnostic active learnable. In *Advances in Neural Information Processing Systems 22*, 2009.

[65] F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.

[66] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *ICDM*. IEEE, 2003.

[67] M. Zinkevich. Online convex programming and generalised infinitesimal gradient ascent. In *Proc. Intl. Conf. Machine Learning*, 2003.